MASTER EN SCIENCES INFORMATIQUES

Le service d'annuaire distribué X.500

Niessen, Fabrice

Award date:
1996

Awarding institution:
Universite de Namur

Facultés Universitaires Notre-Dame de la Paix, NAMUR
Institut d'Informatique
Année académique 1995-1996

Le service d'annuaire distribué X.500

Fabrice NIESSEN

Promoteur : Philippe VAN BASTELAER

Mémoire présenté en vue de l'obtention du grade de
Licencié et Maître en Informatique

Condensé

La norme X.500 décrit un service d'annuaire distribué. Ce service d'annuaire correspond actuellement à un véritable besoin du marché, à l'heure où les outils de communication sont de plus en plus utilisés. La technologie des annuaires est donc une technologie en pleine expansion qui mérite d'être examinée.

Ce mémoire étudie la norme telle qu'elle fut présentée en 1988 et en 1993, et s'intéresse en outre à une implémentation particulière de l'annuaire X.500 : l'annuaire Forum Lookup, développé par la société Telis.

La programmation "multi-thread", appliquée dans le cadre de Forum Lookup, est finalement mise en évidence. Ce mémoire se termine alors par des tests de performance qui montrent la pertinence d'un tel style de programmation.

Abstract

The X.500 standard describes a distributed directory service. This directory service currently corresponds to a real need of the market, when the communication tools become more and more used. The directory technology is thus a fast expanding technology which is worth to be examined.

This work studies the standard such as it was presented in 1988 and 1993, and draws attention to a particular implementation of the X.500 directory : the Forum Lookup directory, developed by the Telis company.

The "multi-thread" programming, used within the framework of Forum Lookup, is finally underlined. This work then finishes by performance tests which show the relevance of such a programming style.

Avant-propos

Réaliser un mémoire sur X.500 a été, pour moi, une expérience très enrichissante. Toutefois, il ne s'est pas fait tout seul et je dois remercier tous ceux qui, de près ou de loin, m'ont aidé dans la rédaction de ce mémoire ainsi que tous ceux qui m'ont permis de progresser quotidiennement lors de mon stage de fin d'études.

Tout d'abord, j'adresse mes remerciements à Jean-Michel Collomb pour sa patience, sa relecture de mes écrits et ses réponses aux nombreuses questions que je lui ai posées. Ses efforts m'ont permis d'essayer de fournir, dans ce mémoire, des informations complètes et exactes.

Ensuite, je remercie mon compatriote Bernard Heuse pour avoir été à l'origine de mon stage et pour m'avoir soutenu durant cette période.

Parmi les employés de Telis, j'ai encore envie de remercier Elizabeth Roudier, Sega Sako, Ascan Woermann, Gérald Lloubès, Pierre Pacchioni, Alexis Karo et Jean-Charles Picard pour leur accueil et leur aide, ainsi que tous les autres employés de Telis que je n'ai pas cités mais qui ne manqueront pas de se reconnaître.

Je remercie aussi tous les auteurs des documents cités en bibliographie, ces derniers ayant constitué la base de la majorité des informations fournies dans ce texte.

Finalement, je tiens à remercier Philippe van Bastelaer pour l'opportunité qu'il m'a donnée de réaliser ce stage et ce mémoire ainsi que pour ses nombreuses et judicieuses critiques.

Préface

Pré-requis

La lecture de ce mémoire suppose, de la part du lecteur, une bonne connaissance des protocoles du monde OSI et du monde TCP/IP ainsi qu'une certaine familiarité avec le langage de programmation C, avec le système d'exploitation UNIX et avec la syntaxe ASN.1.

Organisation du mémoire

Ce mémoire est découpé en trois parties que nous allons brièvement présenter.

La première partie, "Annuaire X.500", aborde les thèmes relatifs à la norme X.500. Elle se compose des chapitres suivants :

- le chapitre 1, "Introduction à X.500", introduit le service d'annuaire X.500 à partir d'autres services d'annuaire existants et présente quelques-uns de ses avantages;
- le chapitre 2, "Organisation de l'annuaire", décrit les éléments de base de l'annuaire, tant au niveau de son contenu qu'au niveau de son fonctionnement;
- le chapitre 3, "DAP-88/93", détaille le service offert par le protocole DAP et essaie d'attirer l'attention sur les plus importantes différences entre la version '88 du protocole et sa version '93;
- le chapitre 4, "Administration de l'annuaire X.500-93", explicite les quatre principales nouveautés introduites dans l'annuaire version '93.

Les thèmes abordés dans la deuxième partie, "Annuaire Forum Lookup", sont relatifs au stage de fin d'études effectué dans la société Telis. Cette partie est structurée de la façon suivante :

- le chapitre 5, "Telis : une société, des produits", présente une vue de la société Telis qui nous a accueilli pendant les six mois qu'ont duré le stage ainsi que trois des produits qu'elle développe;
- le chapitre 6, "SOLO", décrit le service offert par le protocole SOLO, utilisé par l'implémentation (propre à Telis) de l'annuaire X.500;
- le chapitre 7, "Applications de mise en route", expose les deux premières applications que nous avons développées lors de notre stage;
- le chapitre 8, "Programmation multi-thread", présente les fondements théoriques de la programmation "*multi-thread*" sous SunOS 5.0;
- le chapitre 9, "Evolution de l'architecture du DSA", présente les aspects pratiques de la mise en œuvre de la programmation "*multi-thread*" dans cas d'un serveur;
- le chapitre 10, "Conclusion", termine le mémoire en mettant en lumière ses principaux intérêts.

En annexe, le lecteur peut trouver des compléments d'information :

- l'annexe A, "Classes d'objets et types d'attribut", donne la description en ASN.1 des classes d'objets les plus utiles ainsi que des types d'attribut qui leur sont associés;
- l'annexe B, "Tests de performance du DSA multi-thread", est un recueil de feuilles de calcul présentant le détail complet des tests de performance les plus significatifs effectués sur diverses versions du DSA;
- les acronymes constamment utilisés dans le mémoire;
- les références bibliographiques des textes utilisés pour la rédaction du mémoire;
- l'index avec les principaux mots-clé du mémoire.

Conventions typographiques

Afin de faciliter la lecture de ce mémoire, certaines mises en forme typographiques ont été utilisées. Voici leur signification :

- **cette fonte** est utilisée pour la première apparition judicieuse des acronymes ainsi que des termes importants;
- ***cette fonte*** est utilisée pour les termes sur lesquels on désire attirer l'attention;
- *cette fonte* est appliquée aux noms des primitives de service;
- "*cette fonte*" est utilisée pour tous les termes anglais;
- `cette fonte` est employée dans tous les exemples de code, de sortie système ou de saisie utilisateur. Nous l'utilisons aussi dans les exemples d'appels aux primitives de service.

Table des matières

Première partie

l'annuaire X.500

1. Introduction à l'annuaire X.500

1.1 Utilité des annuaires

A l'heure actuelle, les services d'annuaire sont un besoin fondamental tant pour les humains que pour les systèmes de communication informatiques.

Les utilisateurs humains doivent avoir la possibilité de consulter divers détails sur d'autres personnes : par exemple, des numéros de téléphone, de fax ou des adresses postales.

Les systèmes informatiques ont besoin de services d'annuaire pour plusieurs raisons : par exemple, pour permettre des consultations d'adresses pour une variété de services.

1.2 Quelques services d'annuaire actuels

Nous allons examiner ici différents modèles de services d'annuaire [RFC-1309] de manière à réaliser plus loin les divers avantages offerts par X.500.

1.2.1 Service d'annuaire de la compagnie de téléphone

La compagnie de téléphone maintient une liste comportant le nom de ses abonnés, leur numéro de téléphone ainsi que leur adresse. Ces informations sont accessibles à tout utilisateur de deux façons différentes : en appelant le service de renseignements de la compagnie ou en consultant leur annuaire édité chaque année sur papier.

Le service offert par la compagnie téléphonique est néanmoins très limité :

- on ne peut trouver le numéro de quelqu'un que si l'on connaît son nom et la localité où il habite;
- si deux ou plusieurs personnes de la même localité ont le même nom, nous n'avons aucune information supplémentaire pour trouver le bon numéro parmi les différentes propositions;
- l'annuaire imprimé sur papier peut contenir des informations qui sont incorrectes depuis presque un an tandis que le service de renseignements peut fournir des informations dépassées depuis plus de deux semaines;
- pour obtenir des informations sur des habitants d'un certain pays, il faut appeler le service de renseignements de cette pays : il n'existe pas de numéro unique d'accès au service de renseignements;
- on ne peut acquérir qu'un nombre très limité d'informations par communication avec le service de renseignements. Pour toute demande supplémentaire, il faut obligatoirement les rappeler.

1.2.2 Service d'annuaire finger

Le protocole **finger**, implémenté sur la plupart des machines UNIX qui composent le réseau Internet, nous autorise à acquérir des informations à propos d'une personne en particulier ou d'un nom d'utilisateur sur une machine hôte qui reconnaît le protocole.

Ce service est invoqué en introduisant `finger` suivi du nom d'utilisateur de cette personne et de la machine à laquelle elle a l'habitude de se connecter :

```
finger heuse@sophia.telis-sc.fr
```

Un certain ensemble d'informations nous est alors retourné :

```
Login name: heuse                       In real life: HEUSE Bernard
Directory: /home/heuse                  Shell: /bin/csh
On since Apr 25 13:28:41 on pts/31 from toscana:0.0
15 seconds Idle Time
No unread mail
Project: Working on X.500
Plan:
Write many books, become famous
```

`Finger` souffre aussi de différentes limitations :

- il faut connaître le nom de la machine hôte à interroger pour trouver des renseignements sur une personne donnée;
- cette méthode ne permet de localiser des individus que sur des machines qui reconnaissent le protocole `finger` (c'est-à-dire principalement sur des machines UNIX);
- pour des raisons de sécurité, il arrive souvent que `finger` ne soit pas installé sur les machines;
- il n'est par exemple pas possible de trouver toutes les personnes de "`sophia.telis-sc.fr`" travaillant sur X.500 bien que leur activité soit une information qui nous est retournée.

Par conséquent, `finger` a une utilité très limitée.

1.2.3 Service d'annuaire whois

L'utilitaire **whois** (disponible sur une large variété de systèmes qui composent le réseau Internet) travaille en questionnant une base de données ***centralisée*** aux Etats-Unis. Cette base de données contient principalement des informations sur les gens et les équipements.

Une requête ressemble typiquement à :

```
whois heuse
```

Elle retourne l'information suivante :

```
Heuse, Bernard (BH377)                  heuse@SOPHIA.TELIS-SC.FR
  Telis
  Les Algorithmes
  Pythagore A
  route des Lucioles
  06560 Valbonne
```

```
FR
+33-93-95-40-20

Record last updated on 23-Dec-95.
```

La base de données `whois` est suffisamment grande et étendue pour présenter la plupart des défauts d'une grosse base de données centralisée :

- puisque la base de données est maintenue sur une seule machine, un goulot d'étranglement au niveau du processeur implique des temps de réponse élevés lors des périodes de grande activité, même si la plupart des requêtes ne sont pas liées entre elles;
- la base de données étant maintenue sur une seule machine, une limite de la capacité de stockage oblige les administrateurs à restreindre sévèrement la quantité des informations que chaque entrée de la base de données peut contenir;
- tous les changements à effectuer sur des entrées de la base de données doivent être envoyés à l'administrateur qui devra ensuite les introduire dans la base, ce qui augmente à la fois la durée de la période pendant laquelle l'information n'est pas à jour et la probabilité d'une erreur de transcription.

1.2.4 Service d'annuaire DNS

DNS (*"Domain Name System"*) est utilisé dans le réseau Internet pour garder la trace des correspondances entre adresses IP et noms d'hôte. Cette technologie est, à l'heure actuelle, la plus proche de X.500.

Le principe de base est que, à l'enregistrement de chaque domaine, un ou plusieurs serveurs de noms sont identifiés pour ce domaine. Chacun de ces serveurs de noms doit être capable de fournir la correspondance entre adresse IP et nom d'hôte pour chaque machine du domaine. Ainsi, le service est fourni de manière ***distribuée***, ce qui assure des consultations très rapides dans l'espace des noms de domaine. Il est même possible de diviser un domaine en sous-domaines, avec un serveur de noms différent pour chaque sous-domaine.

Bien que, dans la plupart des cas, on utilise DNS sans en être conscient, il est possible d'interroger interactivement DNS avec l'utilitaire **`nslookup`**. Voici un exemple de dialogue avec `nslookup` :

```
> info-st22.info.fundp.ac.be
Server:  ns.fundp.ac.be
Address:  138.48.4.4

Name:  info-st22.info.fundp.ac.be
Address:  138.48.5.157
```

La première ligne de ce dialogue est celle introduite par l'utilisateur : elle constitue une demande de renseignements sur l'adresse IP de la machine "`info-st22.info.fundp.ac.be`". La réponse retournée par `nslookup` (les quatre dernières lignes) se décompose en deux parties :

- le nom logique et l'adresse IP du serveur de noms qui donne l'information;

- le nom logique et l'adresse IP de la machine à propos de laquelle nous avons interrogé `nslookup`.

Comme nous venons de le voir, on peut déterminer explicitement l'adresse IP associée à un hôte donné.

Avec DNS, on peut aussi déterminer le nom d'un hôte en fonction de son adresse IP. Pour cela, il suffit de changer le type des informations qui seront retournées par le serveur et d'envoyer une requête où l'on inverse les octets composant l'adresse IP et où on les suffixe par le terme "`.in-addr.arpa.`" :

```
> set querytype=ptr

> 157.5.48.138.in-addr.arpa.
Server:  ns.fundp.ac.be
Address:  138.48.4.4
157.5.48.138.in-addr.arpa       host name = info-st22.info.fundp.ac.be
```

Toutefois, DNS possède aussi certaines limitations :

- DNS ne manipule que des informations sur les adresses IP et sur les noms logiques des machines;
- à l'heure actuelle, DNS a des capacités de recherche très limitées, ce qui le rend incapable de fournir un service d'annuaire substantiel.

1.3 Service d'annuaire X.500

X.500 est une norme conçue pour fournir le service d'annuaire qui consiste à donner, aux utilisateurs humains et aux applications informatiques, un accès via une interface standard à un ensemble d'annuaires distribués et interconnectés entre eux.

Ce véritable service d'annuaire a été standardisé et publié au titre des recommandations **CCITT** (**Comité Consultatif International pour le Télégraphe et le Téléphone**) X.500 / **ISO** ("*International Standards Organization*") 9594 de 1988. Depuis lors, une deuxième version a été publiée : il s'agit de la version '93 qui complète ou précise certaines recommandations de la version '88. Une troisième version du standard est actuellement en cours de préparation (X.500-96).

La norme X.500 est en passe de devenir très populaire au niveau international car elle n'a aucun concurrent à son niveau. De plus, une grande quantité d'implémentations du service d'annuaire X.500 existent déjà sur le marché. Dans la deuxième partie de ce mémoire, nous nous attarderons d'ailleurs sur l'une de ces implémentations : l'annuaire Forum Lookup développé par la société Telis.

Les principales caractéristiques du service d'annuaire X.500 sont les suivantes [RFC-1308] :

- maintenance décentralisée :
  Chaque site disposant d'une partie de l'annuaire X.500 est responsable de ses données de l'annuaire. Ainsi, les mises à jour peuvent être faites instantanément;

- annuaire normalisé :
  L'accès aux informations se fait par des protocoles normalisés (protocoles DAP/DSP que nous décrirons plus tard);
- possibilités de recherche évoluées :
  Les requêtes des utilisateurs peuvent tirer profit des puissantes possibilités de recherche (filtrage des données, utilisation d'expressions booléennes);
- cadre d'informations structuré :
  X.500 définit un cadre d'informations de base tout en autorisant des extensions locales. Ce cadre d'informations est aussi flexible : de nouveaux types d'information (sons, photos, voix, ...) peuvent être enregistrés dans l'annuaire en fonction des besoins.

2. Organisation de l'annuaire

2.1 Introduction

Appréhender l'annuaire X.500 dans son ensemble est assez complexe. Dès lors, pour nous faciliter la tâche, nous allons l'examiner sous différents points de vue.

Pour nous aider à comprendre à quoi ressemble l'information, comment elle est distribuée et manipulée, nous verrons quatre modèles qui présenteront à chaque fois une vue simplifiée d'un seul aspect de l'annuaire ([ROSE-93a] et [CHAD-94]).

Le ***modèle de l'information*** présente une vue de l'information stockée dans l'annuaire, telle qu'elle est vue par un utilisateur normal. Dans ce modèle, l'aspect distribué de l'annuaire est ignoré. On considère juste qu'une grande quantité de données est détenue dans l'annuaire et que tous les utilisateurs peuvent y accéder, pourvu qu'ils en aient le droit.

Le ***modèle du fonctionnement*** s'applique à analyser les interactions entre les différents composants informatiques (qui fournissent le service d'annuaire) pour accéder à l'information distribuée.

Le ***modèle de l'autorité*** aborde la façon dont sont gérées les bases de données locales.

Le ***modèle de la sécurité***, enfin, décrit brièvement les aspects d'authentification et de contrôle d'accès.

2.2 Modèle de l'information

2.2.1 Base de données

L'ensemble complet des informations auxquelles l'annuaire assure l'accès est appelé **base de données de l'annuaire** ("*Directory Information Base*" ou **DIB**).

2.2.2 Entrée

La base de données est constituée d'**entrées**, chaque entrée représentant un **objet** "intéressant" (pour les utilisateurs de l'annuaire) du "monde réel" (par exemple : une personne, une organisation, un serveur, une machine ou un réseau d'ordinateurs).

Remarquons que des objets du monde réel peuvent être représentés par plusieurs entrées dans la base de données de l'annuaire. Ainsi, une personne réelle peut être représentée dans l'annuaire à la fois par une entrée de type "résident" et par une entrée de type "employé".

2.2.3 Attribut

Chaque entrée est composée d'un ensemble d'**attributs** qui contiennent une partie des informations sur l'objet considéré.

Un attribut est composé d'un type et d'une ou plusieurs valeurs.

Le **type d'attribut** identifie :

- la nature de l'information (sémantique) contenue dans l'attribut;
- la syntaxe de l'attribut, qui décrit :
  - ♦ le type des données qu'il contient (nombre entier, chaîne de caractères, adresse postale, ...),
  - ♦ ses caractéristiques de comparaison (comparaison par égalité, sous-chaîne, ...);
- son caractère mono-valué ou multi-valué.

Chaque **valeur d'attribut** représente un fait sur l'objet représenté par l'entrée. Tout attribut peut avoir une valeur dite **spécifique** si elle contribue au nom de l'objet dans la DIB (ceci sera expliqué un peu plus loin).

La Figure 1 [ROSE-93a] illustre la constitution de la DIB à la lumière des définitions que nous venons de considérer.

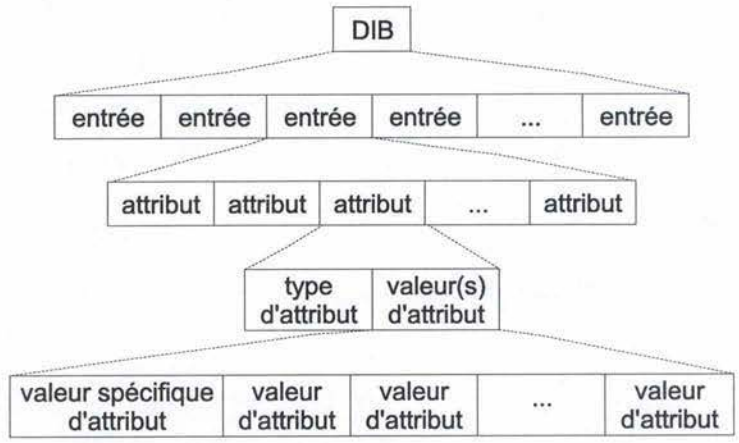


Figure 1 - Constitution de la base de données

On appelle **assertion de valeur d'attribut** ("*Attribute Value Assertion*" ou **AVA**) un couple "type d'attribut = valeur d'attribut".

2.2.4 Classe d'objets

Les objets qui ont des caractéristiques communes sont identifiés par leur **classe d'objets**. Pour chaque classe d'objets, on définit un ensemble d'attributs obligatoires et un ensemble d'attributs facultatifs pour les entrées qui leur correspondent.

La définition de classes d'objets est basée sur la notion d'héritage. Cela signifie qu'une classe d'objets peut être définie comme une sous-classe d'une classe d'objets précédemment définie avec des raffinements supplémentaires. En tant que sous-classe, la nouvelle classe "hérite" alors de tous les attributs obligatoires et facultatifs de sa super-classe en supplément aux siens.

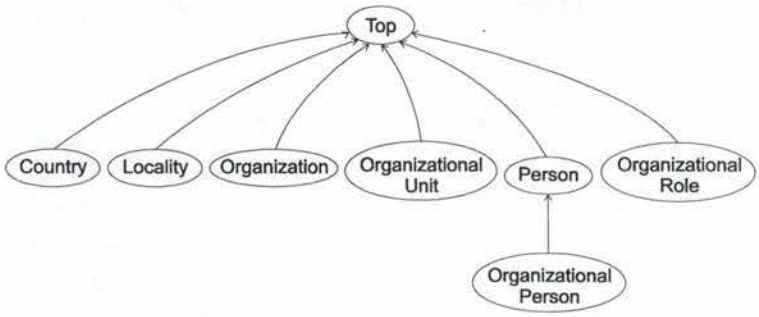


Figure 2 - Hiérarchie des classes d'objets

La Figure 2 montre une partie de la hiérarchie des classes d'objets définies dans X.500. La classe d'objets la plus générique s'appelle "*Top*". Les classes "*Country*", "*Locality*", "*Organization*", "*Organizational Unit*", "*Person*" et "*Organizational Role*" sont des sous-classes de "*Top*". La dernière classe de cette figure, "*Organizational Person*" dérive quant à elle de la classe "*Person*". Le lecteur intéressé trouvera en annexe (annexe A) la définition de ces classes d'objets.

Chaque entrée de l'annuaire contient un attribut particulier, sa **classe** (attribut "*Class*"), qui détermine le type d'objet qu'elle représente. Cet attribut est multi-valué car, si la classe d'objets d'une entrée est une sous-classe d'une super-classe plus générique, alors la chaîne entière des super-classes doit se trouver dans l'attribut "*Class*". Les valeurs de cet attribut déterminent quels attributs l'entrée doit contenir et quels attributs elle peut contenir.

2.2.5 Organisation de la base de données

Les entrées enregistrées dans la DIB sont organisées hiérarchiquement (en structure d'arbre) selon leur classe. La DIB peut être donc être représentée par un arbre, appelé **arbre d'informations de l'annuaire** ("*Directory Information Tree*" ou **DIT**), dans lequel chaque noeud représente une entrée de l'annuaire. Cet arbre fait l'objet de la Figure 3 [CHAD-94].

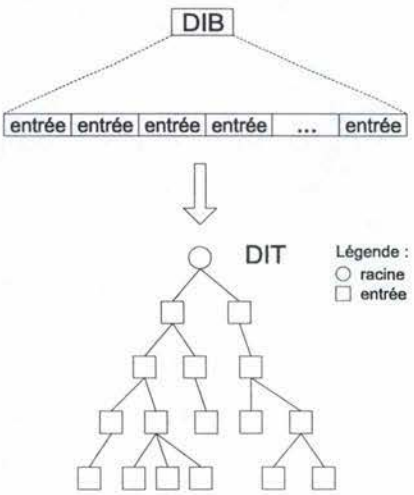


Figure 3 - Organisation hiérarchique des entrées de la DIB

La DIB et le DIT sont simplement des perspectives différentes des entrées contenues dans l'annuaire : le DIT impose une relation hiérarchique entre les entrées alors que la DIB n'a pas ce concept d'entrées associées.

Dans le cas de Forum Lookup (annuaire d'entreprise), la structure du DIT correspond à une division organisationnelle du monde en ***pays*** (classe "*Country*"), ***régions*** (classe "*Locality*"), ***organisations*** (classe "*Organization*"), ***unités organisationnelles*** (classe "*Organizational Unit*"), ***personnes organisationnelles*** (classe "*Organizational Person*") et ***rôles organisationnels*** (classe "*Organizational Role*"). Ce découpage est illustré par la Figure 4 [FORU-95e].

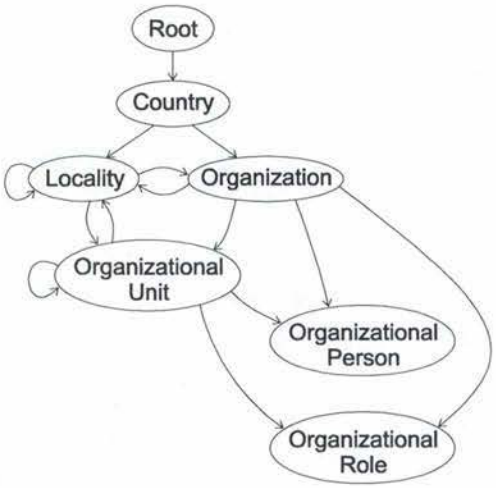


Figure 4 - Structure du DIT de Forum Lookup

La **structure du DIT** contrôle l'organisation hiérarchique des entrées selon leur classe. Ainsi, sous la racine du DIT, ne doit-on trouver que des entrées de classe "*Country*". Sous ces entrées, on peut trouver indifféremment des entrées de classe "*Locality*" ou de classe "*Organization*". De façon similaire, le lecteur pourra aisément continuer l'interprétation de la Figure 4.

2.2.6 Nommage des entrées

Chaque entrée répertoriée dans l'annuaire a un **nom spécifique** ("*Distinguished Name*" ou **DN**) qui permet de la distinguer de manière unique de toutes les autres. Le DN désigne l'endroit de l'annuaire où est stockée toute information sur un objet donné.

Le DN de chaque entrée est construit en prenant le DN de l'entrée parent dans le DIT et en le suffixant par un **nom spécifique relatif** ("*Relative Distinguished Name*" ou **RDN**) qui permet d'identifier l'entrée par rapport à ses frères.

Le RDN d'une entrée est constitué par l'ensemble des assertions de valeur d'attribut qui ont une valeur spécifique et qui ne sont pas contenues dans l'entrée parent. Bien que cela soit rarement le cas, il peut donc arriver qu'un RDN soit construit à partir de plusieurs AVA.

La seule entrée qui n'a pas de parent est la racine qui, en plus, n'a pas de RDN. Le DN de cette entrée est donc vide. Cette entrée, qui ne correspond à aucun objet du monde réel, n'est là que pour garantir l'unicité des DN au sein de l'annuaire.

La définition d'un DN étant une définition récursive, un DN s'exprime en fait comme une liste de RDN. Cette liste ne peut pas être infinie vu que l'entrée racine (supérieure à toutes les autres) n'a pas de parent.

Les attributs utilisés dans le RDN des entrées sont les suivants :

- "*CountryName*" ou **C** pour les entrées de type "*Country*";
- "*LocalityName*" ou **L** pour les entrées de type "*Locality*";
- "*OrganizationName*" ou **O** pour les entrées de type "*Organization*";
- "*OrganizationalUnitName*" ou **OU** pour les entrées de type "*Organizational Unit*";
- "*CommonName*" ou **CN** pour les entrées de types "*Organizational Person*" ou "*Organizational Role*".

Pour illustrer toutes ces définitions, nous allons montrer un fragment de la base de données au format texte concernant la société Telis.

Bien que la notation textuelle utilisée pour écrire des entrées ne soit pas importante, il est important d'avoir une notation concise.

Dans Forum Lookup, une entrée est représentée par un couple "DN = liste d'attributs".

Un DN est écrit comme une liste de RDN séparés par un point-virgule (";") avec le RDN le plus significatif à gauche. Cette liste est délimitée à gauche par le signe inférieur ("<") et à droite par le signe supérieur (">").

La liste d'attributs est délimitée à gauche par le signe inférieur et à droite par le signe supérieur. Les attributs sont séparés par un point-virgule.

Tous les attributs sont composés d'une chaîne de caractères se référant au type d'attribut, d'un signe égal et d'une liste des valeurs (séparées par une virgule).

Voici donc, comme exemple, une liste d'entrées contenues dans un fragment de la base de données de Telis :

```
<C=FR;O=E3X> = <                                          <-- DN de l'entrée
Class= organization;                                      \
FunctionsList="Ingenieur";                                 \  attributs
LocalityList="Valbonne", "Sophia-Antipolis";               /  de l'entrée
L= "Sophia-Antipolis", "Valbonne"                         /
>


<C=FR;O=E3X;OU=OSI> = <
Class= organizationalUnit;
L= Sophia-Antipolis, Valbonne;
Category= "Software Engineering and formation";
Postadd= <"Les Algorithmes";"Pythagore A";"route des Lucioles";"06560 Valbonne">;
Manager = <C=FR;O=E3X;OU=OSI;CN="Sega Sako"> ;
Secretary = <C=FR;O=E3X;OU=OSI;CN="Nathalie Michaudet">;
Desc="Centre de R&D du departement RCE";
```

```
Email= "sophia.telis-sc.fr";
Phone= "+33 93.95.40.00"
>


<C=FR;O=E3X;OU=OSI;CN="Bernard Heuse"> = <
Class= organizationalPerson;
Email= "heuse@sophia.telis-sc.fr";
Title= Ingenieur;
Postadd= <"Les Algorithmes";"Pythagore A";"route des Lucioles";"06560 Valbonne">;
Activities = X.500;
L= Sophia-Antipolis, Valbonne;
Manager = <C=FR;O=E3X;OU=OSI;CN="Ascan Woermann">;
Secretary = <C=FR;O=E3X;OU=OSI;CN="Nathalie Michaudet">;
Phone= "+33 93.95.40.20"
>


<C=FR;O=E3X;OU=OSI;CN="Jean-Michel Collomb"> = <
Class= organizationalPerson;
Email= "collomb@sophia.telis-sc.fr";
Title= Ingenieur;
Activities = X.500;
L= Sophia-Antipolis, Valbonne;
Manager = <C=FR;O=E3X;OU=OSI;CN="Ascan Woermann">;
Secretary = <C=FR;O=E3X;OU=OSI;CN="Nathalie Michaudet">;
Postadd= <"Les Algorithmes";"Pythagore A";"route des Lucioles";"06560 Valbonne">;
Phone= "+33 93.95.40.65"
>
```

La Figure 5 montre le fragment du DIT correspondant à la DIB donnée précédemment en exemple à laquelle les entrées `<>`, `<C=fr>`, `<C=be>`, `<C=fr;O=inria>`, `<C=be;O=fundp>`, `<C=fr;O=e3x;OU=R&D>` et `<C=be;O=fundp;OU=info>` ont été ajoutées.

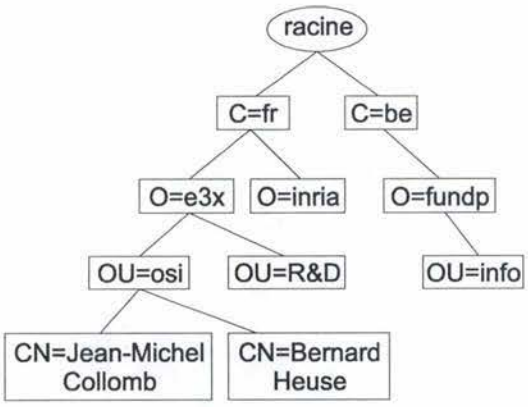


Figure 5 - Fragment du DIT

2.2.7 Alias

Certaines entrées du DIT sont appelées des entrées alias. Une **entrée alias** contient un pointeur vers une autre entrée (c'est-à-dire, pour être précis, qu'elle contient l'attribut mono-valué "*aliasedObjectName*" dont la valeur est le DN d'une autre entrée).

La version '88 du standard interdit à une entrée alias de contenir un pointeur vers une autre entrée alias. Cette restriction a été abandonnée dans la version '93.

Le concept d'alias a été introduit pour permettre à une entrée d'être connue sous plusieurs noms différents (noms alternatifs), ce qui est très pratique. Une entrée peut avoir ainsi un ou plusieurs **noms alias**. Elle n'a toutefois qu'un seul DN.

Par exemple, l'entrée `<C=fr; O=e3x; OU=osi; CN="Laurence Dupont">` a trois noms alias différents dans le DIT illustré à la Figure 6 : `<C=fr; O=telis; OU=osi; CN="Laurence Dupont">`, `<C=fr; O=e3x; OU=osi; CN="Laurence Schneider">` et `<C=fr; O=telis; OU=osi; CN="Laurence Schneider">`. Remarquons, au passage, que les entrées alias sont toujours des entrées feuilles du DIT.

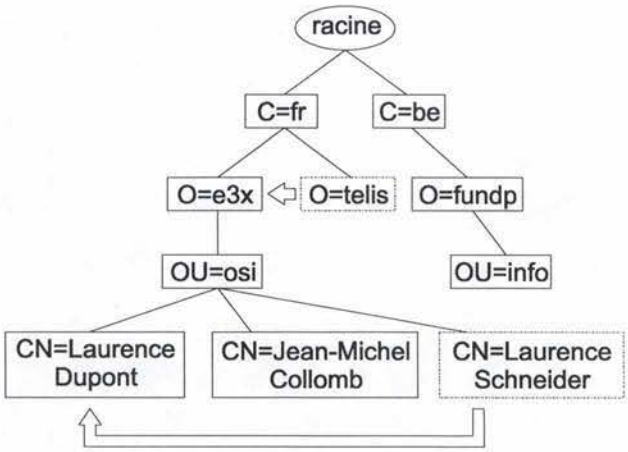


Figure 6 - Entrées alias

Grâce à ce mécanisme, le nom d'une entrée ne doit pas nécessairement être ***unique*** (c'est-à-dire qu'il ne doit pas être le seul nom à identifier l'entrée considérée), il doit seulement être ***non ambigu*** (c'est-à-dire que le nom correspond précisément à une entrée du DIT).

2.2.8 Nom prétendu, résolution du nom et déréférencement des alias

Un utilisateur de l'annuaire donnera généralement un **nom prétendu** comme paramètre d'une requête. Un nom prétendu est syntaxiquement un DN (c'est-à-dire une séquence de RDN) mais il peut être ou ne pas être le nom d'une véritable entrée du DIT.

La **résolution du nom** est une procédure accomplie par l'annuaire pour déterminer, entre autres, si le nom prétendu est valide, c'est-à-dire s'il identifie sans ambiguïté une seule entrée du DIT. La procédure de résolution du nom consiste à faire correspondre séquentiellement chaque RDN du nom prétendu avec un arc dans le DIT, en commençant à partir de la racine et en progressant vers le bas du DIT. Si un alias est rencontré, il est généralement **déréférencé**, c'est-à-dire que la résolution du nom recommence à nouveau à partir de la racine avec le nom (référence) fourni dans l'entrée alias.

2.2.9 Syntaxe UFN

Le nommage des entrées de l'annuaire X.500 est loin d'être convivial. Pour qu'il le devienne, une syntaxe, connue sous le nom de syntaxe **UFN** (*"User-Friendly Naming"*) [RFC-1781], a été mise au point (en dehors de la norme X.500). Nous verrons plus loin que cette syntaxe est admise dans le protocole SOLO (chapitre 6), bien que son utilisation soit interdite dans le protocole DAP (chapitre 3).

Deux aspects sont visés par cette philosophie :

- l'utilisation d'une notation plus pratique (que les DN) pour un utilisateur humain;
- la permission d'utiliser une notation incomplète lorsqu'il n'y a pas d'ambiguïté.

Puisque la syntaxe UFN relâche certaines contraintes de représentation des DN, un nom représenté selon cette syntaxe ne peut pas être automatiquement mis en correspondance avec le ou les DN auxquels il correspond sans consulter l'annuaire.

Nous allons considérer maintenant le DN `<C=fr; O=e3x; OU=osi; CN="Ascan Woermann">` et voir plusieurs façons de le représenter selon la syntaxe UFN :

- omission des types d'attribut : `<Ascan Woermann, osi, e3x, fr>`;
- abréviation : `<Ascan Woermann>`;
- omission d'un composant : `<Ascan Woermann, e3x, fr>`;
- approximation : `<Ascan Worman, osi, e3x, fr>`;
- nom convivial de pays : `<Ascan Woermann, osi, e3x, France>`.

Lorsque les types d'attribut sont omis, le schéma implicite `(CN, OU*, O, C)` est utilisé pour les déduire (l'étoile "`*`" signifiant ici qu'il peut y avoir plusieurs unités organisationnelles composant le nom d'une entrée).

2.2.10 Schéma d'annuaire

Le **schéma d'annuaire** est l'ensemble des définitions et des contraintes permettant aux administrateurs de l'annuaire de contrôler (dans les portions du DIT qu'ils administrent) les informations que les utilisateurs sont autorisés à enregistrer. Le schéma d'annuaire est défini par l'administrateur de la base lui-même. Il comprend :

- la définition (informelle, nous en discuterons plus loin) de la ***structure du DIT*** :
  La structure du DIT contrôle le ***DIT***;
- la définition des ***classes d'objets*** :
  Les classes d'objets contrôlent les attributs obligatoires et les attributs facultatifs des ***entrées***;
- la définition des ***types d'attribut*** :
  Les types d'attribut contrôlent tous les aspects des ***attributs*** (sémantique, caractère multi-valué ou non, ...);
- la définition des ***syntaxes d'attribut*** :
  Les syntaxes d'attribut contrôlent les ***valeurs*** que les attributs peuvent avoir.

La version '93 du schéma est légèrement différente de la version '88. Ces différences ne nous intéressant pas directement, nous préférons les passer sous silence.

2.2.11 Attributs utilisateur, opérationnels et collectifs

Il y a plusieurs sortes d'attributs enregistrés dans l'annuaire : dans la version '88, nous pouvons distinguer les attributs utilisateur et les attributs opérationnels (bien que le standard ne les distingue pas).

Les **attributs utilisateur** sont transparents aux opérations internes de l'annuaire. En d'autres termes, leur présence ou leur absence dans la DIB n'affecte pas le déroulement des opérations de l'annuaire. L'annuaire fonctionnera tout aussi bien sans eux. Par exemple, l'attribut "*Phone*" est un attribut utilisateur.

Les **attributs opérationnels**, par contre, sont liés aux opérations de l'annuaire lui-même. Ils servent à traiter correctement les requêtes des utilisateurs. Par exemple, les attributs de contrôle d'accès sont des attributs opérationnels. Ces attributs ne sont pas visibles aux utilisateurs "normaux" de l'annuaire.

On réalisa rapidement, après '88, que le modèle d'information présenté jusqu'ici est trop simpliste. D'autres sortes d'attributs doivent pouvoir être enregistrés dans l'annuaire. Ainsi, certains attributs peuvent s'appliquer à toute une série d'entrées.

Par exemple, tous les employés d'une société ont une adresse postale identique, celle de leur entreprise. Il est très inefficace de devoir répéter l'adresse postale dans les entrées de tous les employés (c'est pourtant ce qui est fait dans la base de données de Telis). Idéalement, elle ne devrait être stockée qu'à un seul endroit de l'annuaire avec une indication des entrées sur lesquelles elle s'applique. Quand un utilisateur lit l'entrée d'un employé, cet attribut partagé devrait être retourné avec les autres attributs de l'entrée.

Ces nouveaux types d'information, autorisés dans la version '93, sont appelés **attributs collectifs**. Les attributs collectifs sont lus dans les entrées comme les autres attributs utilisateur mais ils ne peuvent pas être mis à jour comme eux. Ceci est du au fait qu'ils ne sont pas réellement stockés dans l'entrée. Ils sont stockés dans une entrée spéciale, appelée **sous-entrée**.

Les sous-entrées ne sont pas visibles aux utilisateurs normaux : elles ne sont accessibles qu'aux administrateurs. Une sous-entrée contient une description du sous-arbre du DIT auquel ses attributs collectifs s'appliquent.

2.2.12 Hiérarchie d'attributs

Un autre nouveau concept introduit par la version '93 du standard est celui de hiérarchie d'attributs. Il formalise le fait que certains types d'attribut peuvent être similaires entre eux et sont en fait des raffinements d'un type plus général.

Un type d'attribut plus général est dit être un super-type d'un type d'attribut plus spécifique. A l'inverse, un type d'attribut plus spécifique est dit être un sous-type d'un type d'attribut plus générique. Par exemple, les types d'attribut "*PortablePhone*" et "*AfterHoursPhone*" peuvent être des sous-types de "*Phone*".

L'utilisation des sous-types simplifie la spécification des attributs puisqu'il n'est pas nécessaire de spécifier la syntaxe d'attribut pour les sous-types. Ces spécifications sont héritées de la définition du super-type.

Quand on cherche à accéder à un type d'attribut de l'annuaire, il renvoie toutes les valeurs de ce type ainsi que toutes celles de ses sous-types qu'il trouve. L'utilisateur est toujours capable de connaître les types des attributs retournés puisqu'ils sont renvoyés avec les valeurs.

2.3 Modèle du fonctionnement

Le modèle du fonctionnement de l'annuaire décrit les interactions entre les différents composants de l'annuaire.

L'annuaire est composé d'un ensemble d'agents, appelés **agents système de l'annuaire** (*"Directory System Agent"* ou **DSA**), qui coopèrent entre eux pour offrir le service requis par l'utilisateur de telle manière que l'annuaire soit perçu comme accessible à partir de n'importe quel DSA. Chaque DSA est en fait un serveur de données qui ne gère qu'un fragment de la base d'informations mais qui est capable, en contrepartie, de contacter d'autres DSA.

Les agents auxquels accèdent les utilisateurs sont appelés **agents utilisateur de l'annuaire** (*"Directory User Agent"* ou **DUA**); ils constituent l'interface entre les utilisateurs et l'annuaire. Pour offrir le service d'annuaire à l'utilisateur, le DUA établit une connexion avec un DSA et l'interroge suivant le **protocole d'accès à l'annuaire** (*"Directory Access Protocol"* ou **DAP**).

Dans le cas le plus simple, le DSA interrogé possède toute l'information nécessaire pour répondre à la requête et la renvoie alors au DUA. Ainsi, à la Figure 7 [ROSE-93a], le DUA envoie une requête au DSA A. Le DSA A étant capable d'y répondre, il retourne au DUA la réponse à cette requête. On parle de résolution locale.

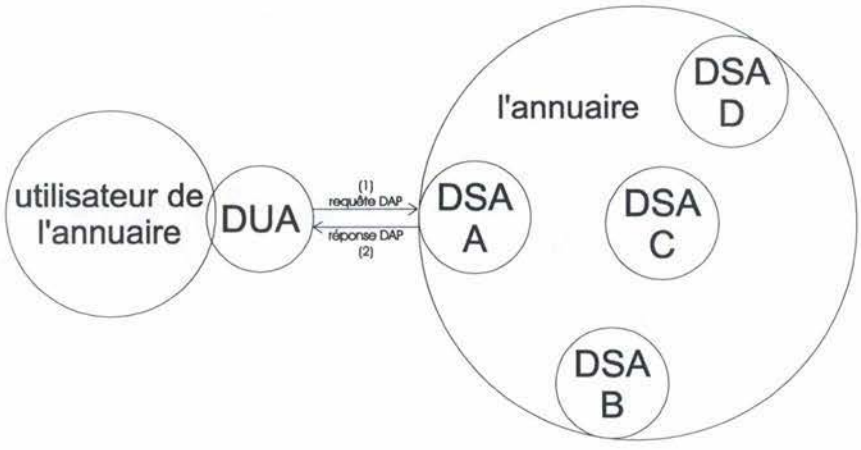


Figure 7 - Résolution locale de la requête

Si le DSA interrogé ne gère pas lui-même les informations qui lui sont demandées, trois principaux cas de figure peuvent être envisagés :

- chaînage de la requête;
- renvoi d'une référence;
- multi-transfert (séquentiel ou parallèle) de la requête.

2.3.1 Chaînage de la requête

Le DSA interrogé transmet la requête à un autre DSA qui, lui, y répondra directement ou transmettra la requête à un tiers. Cette façon de procéder est connue sous le nom de ***chaînage***. La communication entre DSA s'effectue selon le **protocole système de l'annuaire** ("*Directory System Protocol*" ou **DSP**).

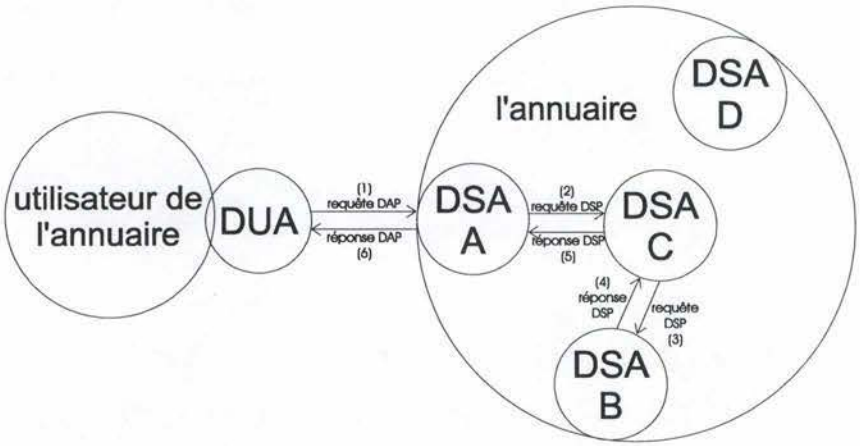


Figure 8 - Chaînage des requêtes

La Figure 8 [ROSE-93a] illustre le cas suivant : le DSA A ne pouvant pas répondre directement à la requête, il la transmet au DSA C. Ce DSA étant aussi incapable de répondre à la requête, il la transmet au DSA B. Le DSA B résout localement la requête et renvoie la réponse au DSA C qui la transmet au DSA A. Le DSA A est alors à même de répondre au DUA.

2.3.2 Renvoi de référence

Le DSA interrogé renvoie au DUA une référence vers un autre DSA (plus "proche" de la réponse) qu'il reste alors à contacter. Ce mode d'interaction est connu sous le nom de ***renvoi de référence***. Remarquons que, dans ce cas, les DUA sont conscients de la distribution des bases d'informations sur plusieurs systèmes (alors que ce fait était ignoré dans le cas précédent).

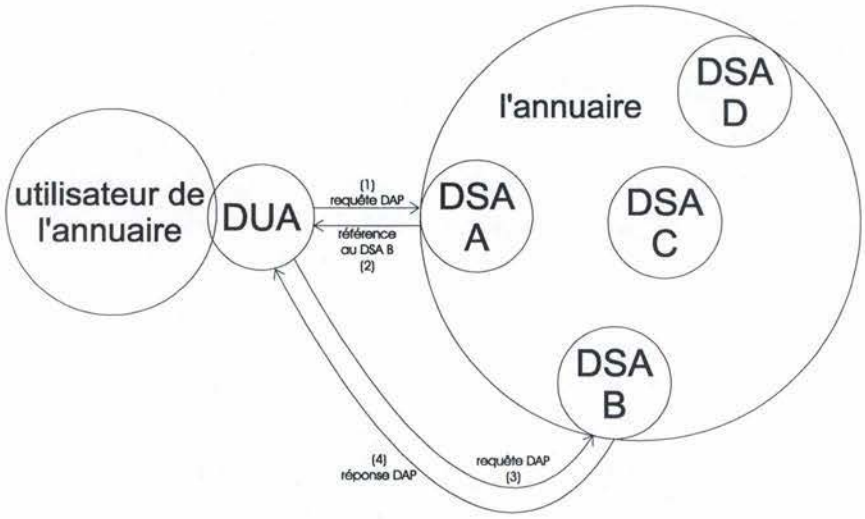


Figure 9 - Renvoi d'une référence vers un autre DSA

La Figure 9 [ROSE-93a] illustre un exemple d'interactions entre le DUA et plusieurs DSA. Le DUA envoie une requête au DSA A. Ce DSA est incapable de répondre directement à la requête mais connaît un DSA qui, lui, est capable d'y répondre. Le DSA A retourne donc une référence vers le DSA B. Le DUA interroge alors le DSA B qui résout localement la requête et retourne au DUA la réponse à celle-ci.

2.3.3 Multi-transfert

Le DSA interrogé envoie la requête à une série d'autres DSA, soit séquentiellement, soit parallèlement. Cette technique porte le nom de ***multi-transfert***. Si le multi-transfert est séquentiel, le DSA attend de recevoir les résultats de la requête chaînée à un des DSA avant de la chaîner vers le DSA suivant. A l'opposé, si le multi-transfert est parallèle, le DSA envoie la requête chaînée à plusieurs DSA simultanément.

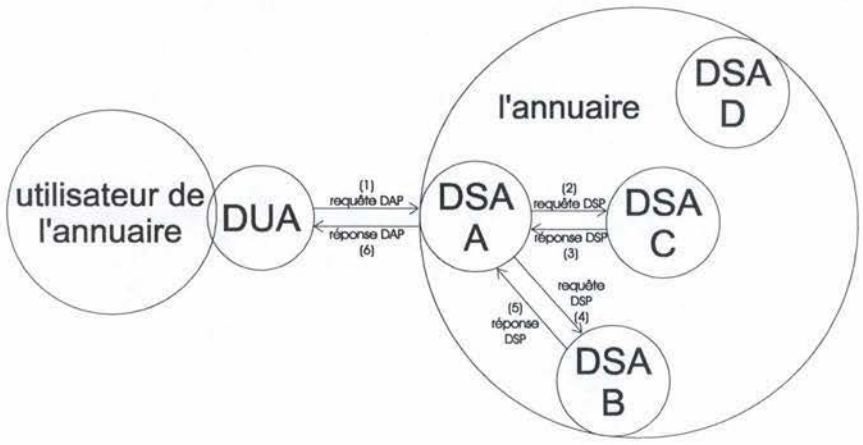


Figure 10 - Multi-transfert séquentiel

La Figure 10 [ROSE-93a] illustre un exemple de multi-transfert séquentiel. Le DUA envoie une requête au DSA A. Ce DSA est incapable de répondre directement à la requête.

Il envoie alors la requête au DSA C. En supposant que le DSA C ne sache pas répondre à la requête, le DSA A, une fois averti de ce fait, va émettre la requête au DSA B, attendre la réponse et la renvoyer au client.

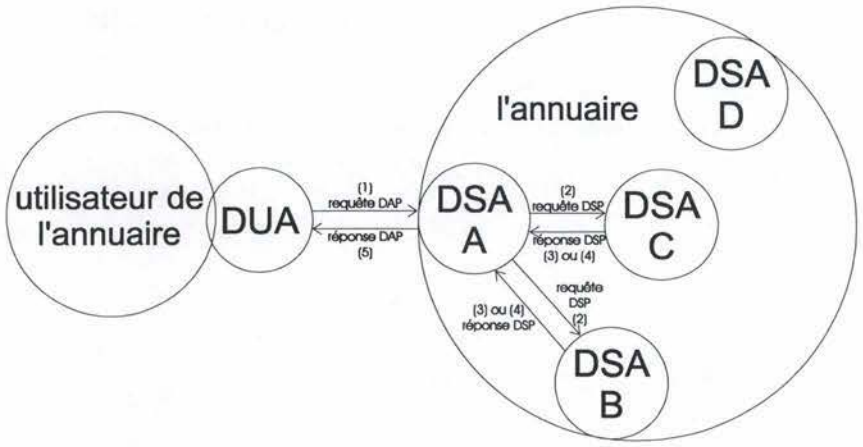


Figure 11 - Multi-transfert parallèle

La Figure 11 [ROSE-93a] illustre un exemple de multi-transfert parallèle. Le DUA envoie une requête au DSA A. Ce DSA étant incapable de répondre directement à la requête, il envoie la requête simultanément au DSA B et au DSA C. Une fois qu'il aura reçu la réponse à sa requête, le DSA A pourra la retourner au DUA.

2.4 Modèle de l'autorité

Le modèle de l'autorité décrit la correspondance entre des portions du DIT et des DSA. Ce modèle s'intéresse donc aussi aux aspects de réplication des données (que nous aborderons plus spécifiquement au point 4.2).

Un **domaine de gestion de l'annuaire** (*"Directory Management Domain"* ou **DMD**) définit une portion du DIT et la manière dont elle est gérée. Il est composé de :

- un ou plusieurs DSA, qui détiennent collectivement la portion du DIT;
- zéro, un ou plusieurs DUA;
- une définition du comportement externe du DMD, établissant comment les multiples DSA dans un DMD doivent être vus de l'extérieur du DMD.

2.5 Modèle de la sécurité

Le modèle de sécurité de l'annuaire décrit le service en termes d'authentification et de contrôle d'accès.

Une politique d'**authentification** est utilisée pour identifier à la fois les DSA et les utilisateurs de l'annuaire. Le but d'une telle politique est de définir des mécanismes par lesquels une entité application est identifiée par le système. Dans l'annuaire, il y a trois types d'authentification :

- aucune authentification;
- authentification simple (basée sur des mots de passe, en clair ou protégés);

- authentification poussée (basée sur la cryptographie de clés publiques).

Une politique de **contrôle d'accès** spécifie à quelles informations les clients peuvent accéder et quels types d'accès leur sont accordés. La politique de contrôle d'accès pour un client est aussi fonction du type d'authentification du client. Cette politique sera brièvement détaillée au point 4.1.

Ces deux politiques vont de pair : il est inutile d'avoir un contrôle d'accès rigoureux si le client peut se faire passer pour quelqu'un d'autre et bénéficier ainsi de ses droits d'accès.

3. DAP-88/93

3.1 Services offerts par l'entité DAP-88/93

Nous allons examiner en détail le service abstrait qui est fourni au DUA par l'entité DAP ([ROSE-93a] et [CHAD-94]). Nous ne nous attacherons donc qu'à décrire les phases *req* et *conf* des primitives de service. La Figure 12 illustre les primitives de service DAP dans l'architecture OSI.

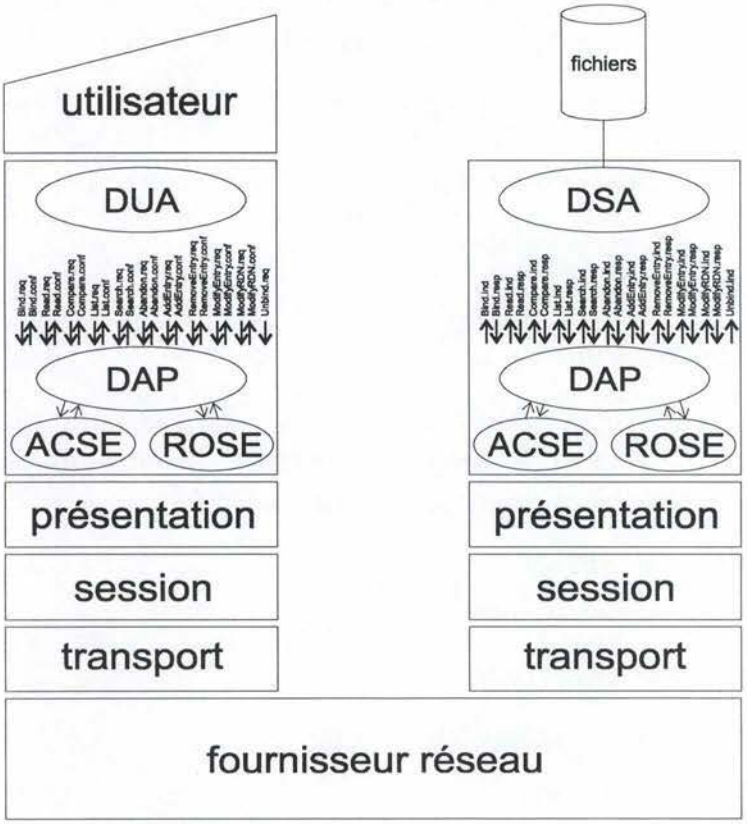


Figure 12 - Architecture et primitives de service DAP

3.1.1 Opérations de connexion / déconnexion

3.1.1.1 Connexion sécurisée

La primitive *Bind* permet à un DUA d'établir une connexion avec un DSA tout en authentifiant son utilisateur pendant l'établissement de la connexion.

Paramètres de *Bind.req* :

- nom du DSA à contacter;
- identité du DUA (optionnelle; valeur par défaut : anonyme), soit simple, soit poussée :
  - ♦ identité simple (DN du DUA et mot de passe, éventuellement protégé),
  - ♦ identité poussée (signature digitale);

- numéro de version du protocole X.500 (optionnel; valeur par défaut : version 1).

Paramètre de *Bind.conf* :

Résultat :

- identité du DSA (optionnelle), soit simple, soit poussée :
  - ♦ identité simple (DN du DSA et mot de passe, éventuellement protégé),
  - ♦ identité poussée (signature digitale).

Erreurs :

- erreur de sécurité, si l'authentification échoue;
- erreur de service, si le service est indisponible, c'est-à-dire si le DSA n'a pas de ressources suffisantes pour fournir un service au client.

3.1.1.2 Déconnexion

La primitive (non confirmée) *Unbind* permet de fermer la connexion établie avec un DSA.

Paramètres de *Unbind.req* :

aucun...

3.1.2 Arguments communs

Toutes les opérations (excepté *Bind*, *Unbind* et *Abandon*, que nous décrirons dans la suite du chapitre) ont un ensemble de paramètres communs qui sont utilisés pour nuancer leur invocation. Les paramètres communs sont les suivants :

- les contrôles de service;
- les paramètres de sécurité;
- le DN du demandeur (DUA);
- l'état d'avancement de l'opération;
- le nombre de RDN déréférencés;
- les extensions.

Les **contrôles de service** décrivent la qualité du service que l'utilisateur attend. Tous ces paramètres sont optionnels mais ils ont tous une valeur par défaut.

Voici la liste des contrôles de service disponibles sur la plupart des opérations de l'annuaire :

- `options` :
  - ♦ `preferChaining` :
    Cette option établit une préférence pour le chaînage plutôt que pour le renvoi de référence. Par défaut, le chaînage n'est pas préféré par rapport aux références.
  - ♦ `chainingProhibited` :
    Cette option interdit l'utilisation du chaînage. Par défaut, le chaînage peut être utilisé.

- `localScope` :
  Cette option interdit à l'opération d'être exécutée en dehors du DSA ou de l'ensemble des DSA du domaine local. La signification précise de "portée locale" n'étant pas standardisée, c'est à l'administrateur qu'il incombe de configurer son DSA pour répondre de façon appropriée. Par défaut, l'opération peut être exécutée globalement.

- `dontUseCopy` :
  Cette option interdit d'utiliser des copies d'entrées (nous détaillerons ce point dans le chapitre suivant). Par défaut, les copies (complètes) d'entrées sont acceptées.

- `dontDereferenceAliases` :
  Cette option interdit de déréférencer les alias pendant la résolution du nom. Par défaut, les alias peuvent être déréférencés.

- `subentries` ('93) :
  Cette option, qui n'a d'effet que sur les opérations *List* et *Search*, permet d'accéder aux sous-entrées. Par défaut, les sous-entrées sont inaccessibles dans les opérations.

- `copyShallDo` ('93) :
  Cette option indique si l'opération peut se satisfaire d'une copie partielle d'une entrée. Par exemple, certains attributs demandés peuvent être absents de la copie, mais l'opération réussira tout de même si cette option est spécifiée. Par défaut, seules les copies complètes d'entrées sont acceptées.

- `priority` :
  Ce contrôle de service indique la priorité de l'opération (`"low"`, `"medium"` ou `"high"`). La valeur par défaut est `"medium"`.

- `timeLimit` :
  Ce contrôle de service impose un nombre maximum de secondes endéans lesquelles le service doit avoir été fourni. Si cette limite de temps vient à être dépassée, alors des résultats partiels sont retournés (et marqués comme tels). Par défaut, il n'y a aucune limite de temps.

- `sizeLimit` :
  Ce contrôle de service impose un nombre maximum d'entrées qui peuvent être retournées par l'opération. Si cette limite de taille vient à être dépassée, alors des résultats partiels sont retournés. Par défaut, il n'y a aucune limite de taille.

- `scopeOfReferral` :
  Ce contrôle de service permet au demandeur de déterminer la portée qu'il juge significative pour les références vers d'autres DSA. Deux valeurs sont admises : `"dmd"` (références uniquement vers le domaine local) ou `"country"` (références uniquement vers le pays local). Par défaut, le DSA peut retourner des références vers n'importe quel autre DSA.

- `attributeSizeLimit` ('93) :
  Ce contrôle de service impose une taille maximum pour tout attribut (son type et toutes ses valeurs) retourné comme résultat de l'opération. La taille est mesurée en octets dans la syntaxe concrète locale du DSA. Si un attribut excède cette taille, toutes ses valeurs sont omises du résultat et l'utilisateur en est averti. Par défaut, il n'y a aucune limite de taille.

Les paramètres de sécurité et le DN du demandeur sont essentiellement destinés à l'authentification poussée de chaque requête utilisateur.

L'état d'avancement de l'opération et le nombre de RDN déréférencés servent lors du chaînage des arguments. Ils concernent donc la distribution de l'annuaire.

Les extensions concernent les nouveautés du standard '93.

3.1.3 Résultats communs

Les quatre opérations de consultation que nous détaillerons ci-après (*Read*, *Compare*, *List* et *Search*) ont un ensemble commun de paramètres utilisés pour nuancer le résultat. Ces paramètres communs sont les suivants :

- les paramètres de sécurité;
- le DN de l'expéditeur du résultat;
- un "*flag*" de déréférencement d'un alias.

Nous avons précédemment parlé de l'utilité des paramètres de sécurité. Nous ne reviendrons donc pas sur ce point ici.

Le DN de l'expéditeur du résultat sert à l'authentification poussée de l'expéditeur de chaque réponse aux requêtes des utilisateurs.

Le "*flag*" de déréférencement d'un alias sert à savoir si le nom prétendu donné dans la requête par l'utilisateur était ou non un alias.

3.1.4 Opérations de consultation

Quatre opérations de consultation (interrogation) sont disponibles ainsi qu'une opération pour les annuler. Ces opérations retournent de l'information sur le DIT ou sur la DIB.

3.1.4.1 Lecture

La primitive *Read* permet de récupérer les valeurs de certains attributs d'une entrée spécifique de l'annuaire.

Paramètres de *Read.req* :

- nom prétendu d'une entrée;
- sélection des informations de l'entrée à retourner :
  - sélection des attributs à retourner (soit tous les attributs utilisateur, soit les attributs spécifiquement mentionnés),

- ♦ sélection des informations à retourner (soit uniquement les types d'attribut, soit les types et leurs valeurs),
- ♦ sélection des attributs supplémentaires ('93) (soit tous les attributs opérationnels, soit les attributs spécifiquement mentionnés);

- demande des droits de modification de l'utilisateur sur l'entrée ('93) (optionnelle);
- arguments communs.

Paramètres de *Read.conf* :

Résultat :

- informations de l'entrée demandées :
  - ♦ DN de l'entrée (nom prétendu de l'entrée, '93),
  - ♦ "*flag*" indiquant si les informations ont été récupérées de l'entrée (plutôt que d'une copie),
  - ♦ informations demandées (soit un ensemble de types d'attribut, soit un ensemble de types et de valeurs d'attribut),
  - ♦ "*flag*" indiquant si l'information de l'entrée est incomplète ('93);
- droits de modification demandés ('93) (optionnels);
- résultats communs.

Erreur :

- erreur d'attribut, si le DSA ne sait rien retourner de ce que l'utilisateur avait demandé.

3.1.4.2 Comparaison

La primitive *Compare* permet de vérifier si une entrée contient une valeur particulière d'un attribut. Elle laisse ainsi la véritable valeur cachée dans l'annuaire.

Cette primitive est notamment utilisée dans la procédure d'authentification simple pour vérifier la valeur de mots de passe.

Paramètres de *Compare.req* :

- nom prétendu d'une entrée;
- valeur prétendue d'un attribut;
- arguments communs.

Paramètres de *Compare.conf* :

Résultat :

- DN de l'entrée (optionnel);
- résultat de la comparaison (soit "`vrai`", soit "`faux`");
- "*flag*" indiquant si l'information a été récupérée de l'entrée;
- sous-type utilisé dans la comparaison ('93) (optionnel);
- résultats communs.

Erreurs :

- diagnostic d'erreur.

3.1.4.3 Enumération

La primitive *List* permet de faire l'inventaire de tous les noeuds subordonnés à un noeud donné.

Grâce à elle, on peut, par exemple, trouver les noms de tous les employés d'un département donné dans une organisation.

Paramètres de *List.req* :

- nom prétendu d'une entrée;
- demande de résultats paginés ('93) (optionnelle; nous n'en parlerons pas);
- arguments communs.

Paramètres de *List.conf* :

Résultat :

- DN de l'entrée (optionnel);
- ensemble d'informations sur les entrées subordonnées :
    - RDN,
    - "*flag*" indiquant si l'entrée subordonnée est une entrée alias ou pas,
    - "*flag*" indiquant si l'information provient directement de l'entrée subordonnée ou pas;
- raison du résultat partiel, si l'ensemble de subordonnés n'est pas complet (optionnelle);
- résultats communs;
- résultats additionnels provenant d'autres DSA (optionnels).

Erreurs :

- diagnostic d'erreur.

3.1.4.4 Recherche

La primitive *Search* permet de rechercher des informations dans un sous-arbre donné du DIT.

Paramètres de *Search.req* :

- nom prétendu de l'**entrée de base** (entrée à partir de laquelle la recherche doit commencer);
- portée de la recherche :
    - uniquement l'entrée de base, ou
    - uniquement les entrées directement subordonnées à l'entrée de base, ou
    - tout le sous-arbre commençant à l'entrée de base;
- filtre booléen de sélection des entrées de la portée :
    - filtre simple (égalité, sous-chaîne, correspondance approximative, ...), ou
    - filtre complexe (AND, OR, NOT) composé de plusieurs critères de sélection;
- "*flag*" indiquant si les alias de la portée doivent être déréférencés;

- sélection des informations des entrées à retourner :
    - ♦ sélection des attributs à retourner (soit tous les attributs utilisateur, soit les attributs spécifiquement mentionnés),
    - ♦ sélection des informations à retourner (soit uniquement les types d'attribut, soit les types et leurs valeurs),
    - ♦ sélection des attributs supplémentaires ('93) (soit tous les attributs opérationnels, soit les attributs spécifiquement mentionnés);
- demande de résultats paginés ('93) (optionnelle);
- filtre additionnel à utiliser par les DSA-93 dans un environnement mixte, composé de DSA-88 et de DSA-93 ('93) (optionnel);
- "*flag*" indiquant si seules les valeurs satisfaisant le filtre doivent être retournées (éliminant ainsi les autres valeurs d'un attribut multivalué) ('93);
- arguments communs.

Paramètres de *Search.conf* :

Résultat :

- DN de l'entrée de base (optionnel);
- ensemble d'informations sur les entrées satisfaisant le filtre :
    - ♦ DN de l'entrée (nom prétendu de l'entrée, '93),
    - ♦ "*flag*" indiquant si les informations ont été récupérées de l'entrée (plutôt que d'une copie),
    - ♦ informations demandées (soit un ensemble de types d'attribut, soit un ensemble de types et de valeurs d'attribut),
    - ♦ "*flag*" indiquant si l'information de l'entrée est incomplète ('93);
- raison du résultat partiel, si l'ensemble des entrées n'est pas complet (optionnelle);
- résultats communs;
- résultats additionnels provenant d'autres DSA (optionnels).

Erreurs :

- diagnostic d'erreur.

3.1.4.5 Abandon

La primitive *Abandon* permet au DUA d'abandonner n'importe quelle opération de consultation en cours.

Paramètre de *Abandon.req* :

- identifiant d'une opération.

Paramètres de *Abandon.conf* :

Résultat :

- accusé de réception.

Erreurs :

- erreur lors de l'abandon, si l'opération d'abandon échoue.

L'identifiant d'une opération est un paramètre retourné par le **ROSE** (*"Remote Operations Service Element"*) quand l'opération est passée pour le transfert vers le DSA. Il identifie de façon unique l'opération de telle façon que tout résultat ou toute erreur (qui contient l'identifiant) peut être mis en correspondance avec l'opération qui l'a généré.

Pour rappel, l'**ACSE** (*"Association Control Service Element"*) et le ROSE sont deux composants de la couche OSI application qui offrent des services communs à la plupart des applications. L'ACSE est responsable de l'ouverture et de la fermeture de toute connexion tandis que le ROSE, lui, est responsable des interactions requête/réponse.

3.1.5 Opérations de modification

Les opérations de modification (administration) ne sont généralement accessibles qu'à l'administrateur (utilisateur privilégié) de la base. On en recense quatre qui changent la structure du DIT ou modifient l'information contenue dans la DIB.

Les opérations telles qu'elles étaient définies en 1988 avaient certaines limitations. La plupart de ces limitations ont été supprimées dans l'édition de 1993.

3.1.5.1 Ajout d'une entrée feuille

La primitive *AddEntry* permet d'ajouter une entrée feuille au DIT.

Paramètres de *AddEntry.req* :

- DN de la nouvelle entrée (nom prétendu de la nouvelle entrée, '93);
- attributs (types et valeurs) de la nouvelle entrée;
- DSA dans lequel l'entrée doit être ajoutée ('93);
- arguments communs.

Paramètre de *AddEntry.conf* :

Résultat :

- accusé de réception.

Erreur :

- diagnostic d'erreur.

3.1.5.2 Suppression d'une entrée feuille

La primitive *RemoveEntry* permet de supprimer une entrée feuille du DIT.

Paramètre de *RemoveEntry.req* :

- DN d'une entrée;
- arguments communs.

Paramètre de *RemoveEntry.conf* :

Résultat :

- accusé de réception.

Erreur :

- diagnostic d'erreur.

3.1.5.3 Modification du contenu d'une entrée

La primitive *ModifyEntry* permet de modifier le contenu d'une entrée.

Paramètres de *ModifyEntry.req* :

- DN d'une entrée (nom prétendu d'une entrée, '93);
- liste des modifications à effectuer sur l'entrée :
  - ajout d'un attribut,
  - suppression d'un attribut,
  - ajout d'une ou de plusieurs valeurs à un attribut,
  - suppression d'une ou de plusieurs valeurs d'un attribut;
- arguments communs.

Paramètre de *ModifyEntry.conf* :

Résultat :

- accusé de réception.

Erreur :

- diagnostic d'erreur.

On peut spécifier autant de modifications que l'on veut dans une seule opération mais elles doivent toutes réussir, sinon aucune ne sera effectuée. Autrement dit, l'opération est atomique.

Remarquons que l'on ne peut pas modifier le RDN d'une entrée avec cette primitive.

3.1.5.4 Modification du nom d'une entrée

La primitive *ModifyRDN* permet de modifier le RDN d'une entrée feuille ('88). Cette primitive ne permet donc pas de restructurer le DIT puisqu'elle interdit de renommer toute entrée qui n'est pas une entrée feuille.

La portée de cette primitive a été considérablement étendue en '93. Son nom a d'ailleurs été rebaptisé pour l'occasion en *ModifyDN*. *ModifyDN* accepte de renommer les entrées feuilles et celles qui ne le sont pas, pour autant que toutes les entrées affectées par cette modification se trouvent dans le même DSA. Cette primitive permet donc de déplacer des sous-arbres vers de nouvelles positions dans le DIT.

Paramètres de *Modify(R)DN.req* :

- DN de l'entrée à renommer;
- nouveau RDN de l'entrée;
- "*flag*" indiquant s'il faut supprimer l'ancien RDN de l'entrée;
- DN du nouveau parent de l'entrée ('93) (optionnel);
- arguments communs.

Paramètre de *Modify(R)DN.conf* :

Résultat :

- accusé de réception.

Erreur :

- diagnostic d'erreur.

3.1.6 Erreurs

Le standard a essayé de définir toutes les erreurs concevables qui pouvaient se produire en réponse à une requête. Ainsi, les différentes implémentations des DSA peuvent se comprendre entre elles, ce qui est particulièrement important lorsqu'il s'agit de relayer ces divers messages.

Six types d'erreur ont été définis. On va maintenant les examiner chacun à leur tour.

3.1.6.1 Erreurs de nom

Une **erreur de nom** est utilisée pour rapporter un problème avec le nom prétendu donné par l'utilisateur. En particulier, un des attributs des RDN peut être faux ou le nom prétendu peut localiser un alias, ce qui est erroné.

3.1.6.2 Erreurs d'attribut

Une **erreur d'attribut** est générée pendant l'évaluation de l'opération, c'est-à-dire après que la résolution du nom se soit terminée avec succès et que le nom prétendu se soit avéré être valide. Elle est utilisée pour signaler qu'une erreur en relation avec un attribut s'est produite.

3.1.6.3 Erreurs de service

Une **erreur de service** peut être générée par n'importe quel DSA, à n'importe quelle étape du traitement de l'opération. Les erreurs de service décrivent généralement des problèmes de fourniture du service d'annuaire, plutôt que des problèmes dans la requête du client.

3.1.6.4 Erreurs lors de l'abandon

Une **erreur lors de l'abandon** est renvoyée à une opération d'abandon qui n'a pas pu être exécutée avec succès. Dans ce cas, l'opération que l'utilisateur désirait abandonner se terminera tout à fait normalement.

3.1.6.5 Erreurs de mise à jour

Une **erreur de mise à jour** est rapportée par le DSA qui évalue l'opération de mise à jour, après que l'entrée ait été trouvée avec succès.

3.1.6.6 Erreurs de sécurité

Une **erreur de sécurité** peut être retournée par n'importe quel DSA impliqué dans le traitement d'une opération. Vu la nature sensible des politiques de sécurité, certaines erreurs de sécurité donnent très peu d'informations utiles à l'utilisateur.

3.1.7 Autres types de réponse

En plus des erreurs, le standard a défini deux autres types de réponse de l'annuaire à une requête. Nous allons maintenant les détailler.

3.1.7.1 Références

Une **référence** est un ensemble d'informations qui permet de continuer le traitement de la requête de l'utilisateur par un autre DSA. Une référence est retournée par un DSA qui ne veut pas ou ne peut pas chaîner la requête vers un autre DSA.

3.1.7.2 Erreurs d'abandon

Une **erreur d'abandon** n'est pas une erreur, contrairement à ce qu'indique son nom. Elle est retournée lorsqu'une opération de consultation a été abandonnée avec succès sur les instructions de l'utilisateur.

3.2 Protocole

3.2.1 Couches inférieures

Comme nous l'avons précédemment signalé, le DUA utilise les services des éléments ACSE et ROSE. Ils utilisent eux-mêmes les services de la couche présentation.

L'ACSE fournit le service qui associe deux applications distantes ensemble. L'association doit être établie avant de transmettre une requête.

Le ROSE fournit un simple service de type requête/réponse. Le DUA fournit une requête au ROSE. Celui-ci en fait un **PDU** ("*Protocol Data Unit*") correct, l'envoie au DSA et retourne au DUA la réponse quand elle arrive. Toutes les requêtes X.500 passent par le ROSE.

3.2.2 Liste des opérations

L'**ASN.1** ("*Abstract Syntax Notation One*") est utilisé pour décrire les objets de la couche application du modèle OSI. En particulier, il est utilisé pour définir les formats des PDU échangés entre les entités DUA et DSA.

Nous allons donner ici, en ASN.1, la description (version '93) des opérations, de leurs arguments et de leurs résultats. Pour ne pas submerger le lecteur, nous laisserons tomber les types de donnée communs, les opérations *Bind* et *Unbind* ainsi que les erreurs et les autres types de réponse.

Nous allons détailler brièvement le cas de l'opération de lecture de telle façon que le lecteur soit à même d'interpréter les descriptions des autres opérations.

3.2.2.1 Lecture

```
read OPERATION ::= {
    ARGUMENT ReadArgument
    RESULT ReadResult
    ERRORS {
        attributeError |
        nameError |
        serviceError |
        referral |
```

```
        abandoned |
        securityError }
    CODE id-opcode-read }
```

L'opération *Read*, identifiée par la constante `ip-opcode-read`, reçoit en entrée (*Read.req*) un paramètre du type `ReadArgument`. Elle retourne (*Read.conf*) un résultat du type `ReadResult` ou une erreur (du type `attributeError`, `nameError`, `serviceError`, `referral`, `abandoned` ou `securityError`).

```
ReadArgument ::= OPTIONALLY-SIGNED { SET {
    object                  [0] Name,
    selection               [1] EntryInformationSelection DEFAULT { },
    modifyRightsRequest     [2] BOOLEAN DEFAULT FALSE,
    COMPONENTS OF CommonArguments }}
```

Ces quatre paramètres de *Read.req* ont été décrits au point 3.1.4.1.

```
ReadResult ::= OPTIONALLY-SIGNED { SET {
    entry           [0] EntryInformation,
    modifyRights    [1] ModifyRights OPTIONAL,
    COMPONENTS OF CommonResults }}

ModifyRights ::= SET OF SEQUENCE {
    item CHOICE {
        entry       [0] NULL,
        attribute   [1] AttributeType,
        value       [2] AttributeValueAssertion },
    permission [3] BIT STRING { add (0), remove (1), rename (2), move (3) }}
```

Ces trois paramètres de *Read.conf* ont aussi été décrits au point 3.1.4.1.

3.2.2.2 Comparaison

```
compare OPERATION ::= {
    ARGUMENT CompareArgument
    RESULT CompareResult
    ERRORS {
        attributeError |
        nameError |
        serviceError |
        referral |
        abandoned |
        securityError }
    CODE id-opcode-compare }

CompareArgument ::= OPTIONALLY-SIGNED { SET {
    object      [0] Name,
    purported   [1] AttributeValueAssertion,
    COMPONENTS OF CommonArguments }}

CompareResult ::= OPTIONALLY-SIGNED { SET {
    name                Name OPTIONAL,
    matched         [0] BOOLEAN,
    fromEntry       [1] BOOLEAN DEFAULT TRUE,
    matchedSubtype  [2] AttributeType OPTIONAL,
    COMPONENTS OF CommonResults }}
```

3.2.2.3 *Enumération*

```
list OPERATION ::= {
    ARGUMENT ListArgument
    RESULT ListResult
    ERRORS {
        nameError |
        serviceError |
        referral |
        abandoned |
        securityError }
    CODE id-opcode-list }


ListArgument ::= OPTIONALLY-SIGNED { SET {
    object          [0] Name,
    pagedResults    [1] PagedResultsRequest OPTIONAL,
    COMPONENTS OF CommonArguments }}


ListResult ::= OPTIONALLY-SIGNED { CHOICE {
    listInfo SET {
        name                            Name OPTIONAL,
        subordinates                [1] SET OF SEQUENCE {
            rdn                 RelativeDistinguishedName,
            aliasEntry      [0] BOOLEAN DEFAULT  FALSE,
            fromEntry       [1] BOOLEAN DEFAULT  TRUE },
        partialOutcomeQualifier    [2] PartialOutcomeQualifier OPTIONAL,
        COMPONENTS OF CommonResults},
    uncorrelatedListInfo    [0] SET OF ListResult }}


PartialOutcomeQualifier ::= SET {
    limitProblem                    [0] LimitProblem OPTIONAL,
    unexplored                      [1] SET OF ContinuationReference OPTIONAL,
    unavailableCriticalExtensions   [2] BOOLEAN DEFAULT FALSE,
    unknownErrors                   [3] SET OF ABSTRACT-SYNTAX.&Type  OPTIONAL,
    queryReference                  [4] OCTET STRING  OPTIONAL }


LimitProblem ::= INTEGER {
    timeLimitExceeded (0),
    sizeLimitExceeded (1),
    administrativeLimitExceeded (2) }
```

3.2.2.4 *Recherche*

```
search OPERATION ::= {
    ARGUMENT SearchArgument
    RESULT SearchResult
    ERRORS {
        attributeError |
        nameError |
        serviceError |
        referral |
        abandoned |
        securityError }
    CODE id-opcode-search }


SearchArgument      ::=    OPTIONALLY-SIGNED { SET {
    baseObject          [0] Name,
    subset              [1] INTEGER {
        baseObject(0),
```

```
        oneLevel(1),
        wholeSubtree(2)} DEFAULT baseObject,
    filter            [2] Filter DEFAULT  and : { },
    searchAliases     [3] BOOLEAN  DEFAULT TRUE,
    selection         [4] EntryInformationSelection DEFAULT { },
    pagedResults      [5] PagedResultsRequest OPTIONAL,
    matchedValuesOnly [6] BOOLEAN DEFAULT FALSE,
    extendedFilter    [7] Filter OPTIONAL,
    COMPONENTS OF CommonArguments }}


SearchResult ::= OPTIONALLY-SIGNED { CHOICE {
    searchInfo SET {
        name                          Name OPTIONAL,
        entries                   [0] SET OF EntryInformation,
        partialOutcomeQualifier   [2] PartialOutcomeQualifier OPTIONAL,
        COMPONENTS OF CommonResults },
    uncorrelatedSearchInfo        [0] SET OF SearchResult }}
```

3.2.2.5 *Abandon*

```
abandon OPERATION ::= {
    ARGUMENT AbandonArgument
    RESULT AbandonResult
    ERRORS {
        abandonFailed }
    CODE id-opcode-abandon }


AbandonArgument ::= SEQUENCE {
    invokeID [0] InvokeId }


AbandonResult ::= NULL
```

3.2.2.6 *Ajout d'une entrée feuille*

```
addEntry OPERATION  ::= {
    ARGUMENT AddEntryArgument
    RESULT AddEntryResult
    ERRORS {
        attributeError |
        nameError |
        serviceError |
        referral |
        securityError |
        updateError }
    CODE id-opcode-addEntry }


AddEntryArgument ::= OPTIONALLY-SIGNED { SET {
    object          [0] Name,
    entry           [1] SET OF Attribute,
    targetSystem    [2] AccessPoint OPTIONAL,
    COMPONENTS OF CommonArguments}}


AddEntryResult ::= NULL
```

3.2.2.7 *Suppression d'une entrée feuille*

```
removeEntry OPERATION ::= {
    ARGUMENT RemoveEntryArgument
    RESULT RemoveEntryResult
```

```
    ERRORS {
        nameError |
        serviceError |
        referral |
        securityError |
        updateError }
    CODE id-opcode-removeEntry }


RemoveEntryArgument ::= OPTIONALLY-SIGNED { SET {
    object [0] Name,
    COMPONENTS OF CommonArguments }}


RemoveEntryResult ::= NULL
```

3.2.2.8 Modification du contenu d'une entrée

```
modifyEntry OPERATION ::= {
    ARGUMENT ModifyEntryArgument
    RESULT ModifyEntryResult
    ERRORS {
        attributeError |
        nameError |
        serviceError |
        referral |
        securityError |
        updateError }
    CODE id-opcode-modifyEntry }


ModifyEntryArgument ::= OPTIONALLY-SIGNED { SET {
    object     [0] Name,
    changes    [1] SEQUENCE OF EntryModification,
    COMPONENTS OF CommonArguments }}


ModifyEntryResult ::= NULL


EntryModification ::= CHOICE {
    addAttribute       [0] Attribute,
    removeAttribute    [1] AttributeType,
    addValues          [2] Attribute,
    removeValues       [3] Attribute}
```

3.2.2.9 Modification du nom d'une entrée

```
modifyDN OPERATION ::= {
    ARGUMENT ModifyDNArgument
    RESULT ModifyDNResult
    ERRORS { nameError |
        serviceError |
        referral |
        securityError |
        updateError }
    CODE id-opcode-modifyDN }


ModifyDNArgument ::= OPTIONALLY-SIGNED { SET {
    object          [0] DistinguishedName,
    newRDN          [1] RelativeDistinguishedName,
    deleteOldRDN    [2] BOOLEAN DEFAULT FALSE,
```

```
    newSuperior     [3] DistinguishedName OPTIONAL,
    COMPONENTS OF CommonArguments }}

ModifyDNResult ::= NULL
```

4. Administration de l'annuaire X.500-93

La norme '88 était assez silencieuse à propos de l'administration de l'annuaire [ROSE-93a]. La norme '93, elle, détaille les domaines suivants ([CHAD-94] et [WAUG-94]) :

- administration du ***contrôle d'accès***;
- administration de la ***réplication*** des données ("*shadowing*");
- administration du ***schéma***, qui définit la forme des noms d'entrée et la structure de portions du DIT;
- administration des ***informations de connaissance***, qui définit quel DSA contient quelle portion du DIT.

4.1 Contrôle d'accès

4.1.1 Introduction

Le contrôle d'accès décrit dans le standard est basé sur le stockage de listes de contrôle d'accès. Dans ces listes, on trouve les noms des utilisateurs (ou groupes d'utilisateurs), les informations auxquelles ils peuvent accéder (articles protégés) et les droits d'accès (ou permissions) qui leur sont accordés.

4.1.2 Principes de conception

Le standard a dicté cinq principes généraux concernant le contrôle d'accès.

Premièrement, l'utilisateur n'a pas accès à toutes les informations de l'annuaire pour lesquelles il n'a pas reçu d'autorisation préalable. En d'autres termes, aucun droit d'accès n'est accordé par défaut.

Deuxièmement, lors de sa création, on attribue une priorité à chaque élément de contrôle d'accès d'une liste. Lorsqu'il y a contradiction entre deux éléments, seul l'élément qui a la priorité la plus élevée des deux est appliqué (l'autre est ignoré).

Troisièmement, lorsqu'il y a contradiction entre deux éléments de même priorité, l'élément qui identifie le plus spécifiquement l'utilisateur annule l'autre. Un élément qui s'applique à un utilisateur nommé sera donc appliqué de préférence à un élément qui s'applique à un groupe dont l'utilisateur fait partie.

Quatrièmement, lorsqu'il y a contradiction entre deux éléments de même priorité qui spécifient l'utilisateur de la même façon, l'élément le plus spécifique à propos des articles à protéger l'emporte sur l'autre. Un élément qui s'applique à un attribut spécifique d'une entrée sera donc appliqué de préférence à un élément qui s'applique à tous les attributs de cette même entrée.

Enfin, un élément qui interdit un accès l'emporte toujours sur un élément qui l'autorise lorsqu'il y a conflit entre deux éléments de même priorité et de même spécificité.

4.1.3 Fonction de décision

La **fonction de décision du contrôle d'accès** ("*Access Control Decision Function*" ou **ACDF**) est la formalisation des règles qui permettent de décider si un utilisateur peut accéder à un article protégé en fonction des permissions qui lui ont été affectées préalablement.

4.2 Réplication des données

4.2.1 Introduction

La réplication des données entre plusieurs DSA est bénéfique aux utilisateurs pour deux raisons.

D'une part, elle réduit les temps d'accès lors des recherches et, d'autre part, elle augmente la fiabilité du système. En effet, plutôt que d'être centralisées en un point unique, les données peuvent être répliquées de façon à être localisées à proximité des utilisateurs qui en ont besoin. Et, si une donnée est temporairement indisponible sur un DSA, une de ses copies peut être utilisée à sa place.

Il faut toutefois relever plusieurs inconvénients à la réplication des données :

- complexité accrue d'accès aux données, notamment en ce qui concerne l'administration des informations de connaissance (que nous décrirons plus loin) et la nécessité d'éviter des doublons dans le résultat;
- difficulté de maintenir toutes les données répliquées à jour.

4.2.2 Notions de base

La difficulté la plus importante concerne la mise à jour des données répliquées. La méthode adoptée par le standard consiste à désigner une copie comme étant la **copie maître** à laquelle seront toujours envoyées les mises à jour des utilisateurs. Après un certain temps, les mises à jour sont propagées à toutes les autres copies, appelées **copies esclaves**. Alors que certaines copies esclaves peuvent être mises à jour dans les quelques millisecondes suivant la mise à jour de la copie maître, d'autres peuvent n'être mises à jour que quelques jours plus tard. Une inconsistance temporaire des données est donc tout à fait possible. Néanmoins, si un utilisateur pense qu'une copie esclave locale n'est pas à jour, il peut récupérer la copie maître en utilisant le contrôle de service "`dontUseCopy`" dans sa requête de consultation.

Le DSA qui détient la copie maître d'une entrée est connu sous le nom de **DSA maître**. Les DSA qui détiennent les copies esclaves portent le nom de **DSA esclaves**.

4.2.3 "Shadowing" primaire et secondaire

Pour éviter que le DSA maître doive lui-même fournir les mises à jour à tous les DSA esclaves, un DSA esclave peut jouer le rôle de fournisseur de copies vis-à-vis d'autres DSA

esclaves. Ce processus de réplication s'appelle "***shadowing***" **secondaire**. Lorsque le DSA maître s'occupe lui-même des mises à jour, on parle de "***shadowing***" **primaire**. Ainsi donc, un DSA peut obtenir une copie de la copie maître ou des copies esclaves.

Lorsqu'un DSA envoie des mises à jour, il joue le rôle de **fournisseur** de copies. Lorsqu'il en reçoit, on dit qu'il joue le rôle de **consommateur**.

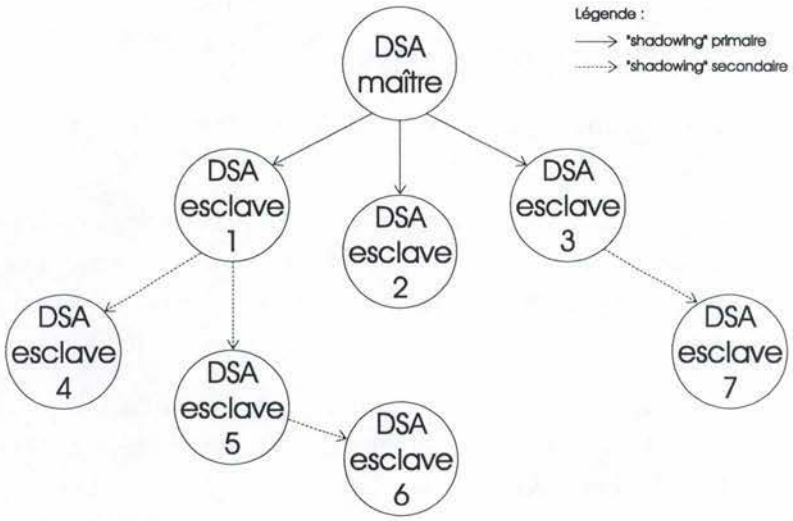


Figure 13 - "*Shadowing*" primaire et secondaire

Le mécanisme de "*shadowing*" est illustré à la Figure 13 [CHAD-94]. Les DSA esclaves 1, 2 et 3 sont fournis en copies par le DSA maître ("*shadowing*" primaire) et sont dès lors des consommateurs de copies. A leur tour, les DSA esclaves 1 et 3 jouent le rôle de fournisseur de copies ("*shadowing*" secondaire) : le DSA esclave 1 fournit les DSA esclaves 4 et 5 et le DSA esclave 3 fournit le DSA esclave 7. Le DSA esclave 5, bien que consommateur vis-à-vis du DSA esclave 1, est aussi fournisseur du DSA esclave 6.

4.2.4 Protocoles

Deux protocoles sont utilisés pour supporter le "*shadowing*". Le premier, le **DOP** ("*Directory Operational binding management Protocol*"), est utilisé pour établir un accord de "*shadowing*" entre deux DSA, le fournisseur et le consommateur. Cet accord décrit l'information à répliquer (attribut, entrée, sous-arbre) et contrôle la fréquence de mise à jour des données répliquées.

Le second protocole, le protocole **DISP** ("*Directory Information Shadowing Protocol*"), est utilisé pour transférer les mises à jour du fournisseur au consommateur.

4.3 Schéma

Parmi les quatre composants du schéma d'annuaire version '88, le seul qui n'est pas formellement défini est la définition de la structure du DIT. Le standard '88 donne bien des suggestions (exemples de structures du DIT) mais précise qu'elles n'ont pas force de loi. En particulier, le schéma '88 ne donne aucune règle sur la façon d'attribuer des noms aux entrées.

Le standard '93, lui, comble ces manques par l'intermédiaire d'un schéma plus complet que le précédent. Le schéma '93 spécifie entre autres quels sont les types d'attribut à utiliser pour former le RDN d'une entrée. Il spécifie aussi quelles entrées peuvent être hiérarchiquement supérieures à d'autres selon leur classe. Par conséquent, le schéma '93 contrôle la façon dont les RDN sont ajoutés ensemble pour former des DN et règle ainsi le problème du nommage.

4.4 Informations de connaissance

Avant d'aborder spécifiquement le concept d'informations de connaissance, décrivons brièvement la façon dont le DIT est distribué entre plusieurs DSA.

Le DIT est partitionné en sous-arbres de taille arbitraire (choix des administrateurs) appelés **contextes de nommage**. Chacun de ces sous-arbres réside en entier dans un DSA, voire plusieurs DSA (en cas de réplication). Pour être distribué sur plusieurs DSA, un sous-arbre doit avoir été préalablement découpé en sous-arbres de plus petite taille.

Le DN de l'entrée qui est à la racine du contexte de nommage est appelé **préfixe du contexte** de nommage. Toutes les entrées d'un contexte de nommage ont évidemment son préfixe comme préfixe de leur DN.

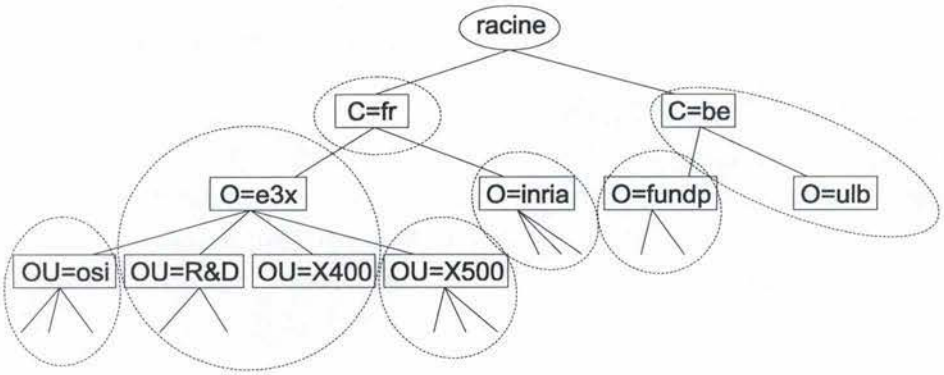


Figure 14 - Contextes de nommage

La Figure 14 [CHAD-94] montre la partition du DIT en sept contextes de nommage dont un ne contient qu'une seule entrée. Leurs préfixes sont `<C=fr>`, `<C=fr; O=e3x>`, `<C=fr; O=e3x; OU=osi>`, `<C=fr; O=e3x; OU=X500>`, `<C=fr; O=inria>`, `<C=be>` et `<C=be; O=fundp>`.

Un DSA détient un nombre quelconque de sous-arbres maximaux. En spécifiant que les sous-arbres doivent être maximaux, on entend qu'un DSA ne peut pas détenir deux sous-arbres verticalement adjacents, mais devra plutôt détenir le sous-arbre maximal qu'ils forment.

Les contextes de nommage sont les unités de distribution du DIT parmi les DSA. Ils possèdent tous des **informations de connaissance**. Ce sont des méta-données (indispensables au fonctionnement distribué de l'annuaire) qui identifient les DSA qui détiennent un contexte de nommage particulier. Les informations de connaissance renseignent donc sur la localisation de toutes les données de l'annuaire, qu'il s'agisse des attributs d'une entrée ou du RDN de ses subordonnés immédiats.

Les informations de connaissance contiennent entre autres :

- le DN et l'**adresse de présentation** (*"Presentation Service Access Point"* ou **PSAP**) d'un DSA distant;
- le nom d'un contexte de nommage (identifié par son préfixe) détenu par le DSA distant;
- leur **catégorie** qui indique si la référence pointe vers la copie maître du contexte de nommage ou vers une copie esclave ('93).

Quatre types d'information de connaissance sont définis dans la version '88 du standard : il s'agit des références supérieures, des références subordonnées, des références subordonnées non spécifiques et des références croisées.

La version '93 a formalisé le modèle utilisé dans la version '88 et a ajouté les références subordonnées immédiates, les références fournisseur et les références consommateur. Les deux derniers types de référence proviennent de la possibilité accordée par la version '93 de répliquer des données.

Les références suivantes peuvent être associées à chaque contexte de nommage.

4.4.1 Référence supérieure

Une **référence supérieure** pointe vers un DSA qui a des connaissances sur des entrées situées dans le haut du DIT, vers la racine.

Quand un DSA a besoin de "se déplacer en haut", il utilise une référence supérieure.

4.4.2 Référence subordonnée

Une **référence subordonnée** pointe vers un DSA qui gère un contexte de nommage situé immédiatement sous le contexte de nommage auquel elle est associée.

La référence subordonnée spécifie le préfixe du contexte de nommage subordonné auquel elle se rapporte. Ce n'est pas le cas pour les références subordonnées non spécifiques que nous détaillons dans la rubrique suivante.

4.4.3 Référence subordonnée non spécifique

Une **référence subordonnée non spécifique** est un type de référence subordonnée qui ne spécifie pas le préfixe du contexte de nommage subordonné auquel elle se rapporte.

Avec ce type de référence, un DSA peut savoir qu'un autre DSA détient des entrées subordonnées à une entrée particulière sans savoir lesquelles.

Lorsqu'un DSA doit "se déplacer en bas" pour résoudre une requête, il doit souvent faire du multi-transfert en présence de telles références. Au contraire, s'il n'y a aucune référence subordonnée non spécifique, le DSA peut directement sélectionner la ou les références subordonnées qui l'intéressent et agir en conséquence.

4.4.4 Référence croisée

Une **référence croisée** pointe vers un DSA qui gère un contexte de nommage quelconque.

Il s'agit d'une optimisation qui permet à un DSA de "se déplacer" sans avoir à utiliser une référence supérieure.

4.4.5 Référence supérieure immédiate

Une **référence supérieure immédiate** pointe vers un DSA qui gère un contexte de nommage situé immédiatement au-dessus du contexte de nommage auquel elle est associée.

Une référence supérieure, elle, pointe vers un DSA qui gère un contexte de nommage plus proche de la racine du DIT, mais pas nécessairement un supérieur immédiat.

4.4.6 Référence fournisseur

Une **référence fournisseur** est détenue par tout DSA consommateur de copies de contextes de nommage. Elle pointe vers le DSA qui fournit la copie.

Rappelons que le fournisseur n'est pas toujours le DSA qui contient la copie maître. Le DSA maître peut néanmoins toujours être trouvé en suivant la chaîne des références fournisseur.

4.4.7 Référence consommateur

Une **référence consommateur** est détenue par tout DSA fournisseur de copies de contextes de nommage. Elle pointe vers un DSA qui consomme la copie.

Puisqu'un fournisseur peut fournir plusieurs consommateurs, il peut avoir plusieurs références consommateur.

Deuxième partie

l'annuaire Forum Lookup

5. Telis : une société, des produits

5.1 Introduction

Dans ce chapitre, nous allons d'abord présenter brièvement la société Telis qui nous a accueilli en stage durant six mois.

Ensuite, nous exposerons trois des nombreux produits qu'elle développe. Le premier produit, l'annuaire Forum Lookup, est le sujet même de la deuxième partie de ce mémoire. Les deux autres produits, le compilateur MAVROS et les couches de communication, sont introduits vu leur importance dans le projet Forum Lookup.

5.2 Présentation de la société

Telis est une filiale du Groupe France Telecom (opérateur français de télécommunications). Avec un chiffre d'affaires de 1.200 MFF et 2.400 employés, Telis est la troisième **SSII** (**Société de Service et d'Ingénierie Informatique**) française sur le marché de l'ingénierie et de l'intégration de systèmes [TS&C-96].

Elle est composée de trois divisions :

- Telis Télécom;
- Telis Ingénierie;
- Telis Systèmes & Communications;

Telis ***Télécom*** réalise les applications métier du Groupe France Telecom et développe son savoir-faire à destination des opérateurs internationaux. Elle réalise un chiffre d'affaires de plus de 500 MFF et emploie plus de 1.000 personnes.

Telis ***Ingénierie*** propose ses prestations d'ingénierie et d'intégration de systèmes aux entreprises et aux administrations. Elle réalise un chiffre d'affaires de plus de 500 MFF et emploie plus de 1.100 personnes.

Telis ***Systèmes & Communications*** (**S&C**) développe son expertise dans le domaine du réseau et de la communication d'entreprise. Elle réalise un chiffre d'affaires de plus de 140 MFF et emploie plus de 220 personnes.

Telis S&C, qui dédie ses développements aux mondes OSI et Internet, est la division dans laquelle nous avons fait notre stage. Cette division est elle-même répartie sur plusieurs sites géographiques en France et compte notamment un centre de **R&D** (**Recherche et Développement**) à Sophia-Antipolis (dans le sud-est).

Quatre équipes travaillent dans ce centre de R&D qui emploie environ 40 personnes. Il s'agit des équipes suivantes :

- l'équipe X.500, qui développe des services d'annuaire;
- l'équipe X.400, qui développe des services d'échange entre personnes;

- l'équipe **EDI** ("*Electronic Data Interchange*"), qui traite les échanges de données informatisés;
- l'équipe "micro", qui s'occupe des portages sur PC et Macintosh.

C'est précisément à l'équipe X.500 de Sophia-Antipolis que nous avons été intégré lors de notre stage.

La liste des travaux qui nous ont été soumis fera l'objet des chapitres 7 et 9.

5.3 Annuaire d'entreprise Forum Lookup v1.1

5.3.1 Présentation générale

L'annuaire Forum Lookup est l'offre produit d'annuaire d'entreprise de Telis S&C. Par annuaire d'entreprise, on entend le fait qu'il permet de localiser directement des personnes au sein d'une entreprise et non au sein d'une localité (comme cela serait le cas pour un annuaire géographique). Ainsi, l'on peut voir l'annuaire Forum Lookup comme étant une bibliothèque dont les travées représentent les pays et dont les rayons désignent les entreprises. Chaque livre serait alors le recueil des personnes travaillant dans un département donné d'une entreprise.

L'annuaire Forum Lookup repose, en outre, sur la norme X.500, ce qui garantit son interopérabilité avec d'autres annuaires.

Enfin, l'annuaire Forum Lookup [TS&C-96] est destiné aux entreprises qui se sont dotées de services d'échange de courrier ou de gestion de leurs informations et pour lesquelles connaître les coordonnées d'un correspondant ou la localisation d'une information devient incontournable. Il inclut tous les produits (des postes de travail aux serveurs) qui sont nécessaires à la mise en place d'un service d'annuaire répondant aux besoins d'une telle entreprise. Forum Lookup sert donc de base pour les solutions propriétaires de ses clients.

5.3.2 Présentation technique

L'annuaire Forum Lookup v1.1 est une implémentation particulière, parmi d'autres, de la norme X.500 telle qu'elle fut définie en 1988. Tout ce qui a été dit dans les chapitres précédents à propos de X.500-88 s'applique donc dans le cadre de Forum Lookup.

Dans Forum Lookup, les postes client utilisent le protocole simplifié **SOLO** ("*Simple Object LOokup*") pour consulter l'annuaire. Ce protocole, actuellement en cours de standardisation, n'est qu'un protocole de ***consultation*** de l'annuaire en mode ***texte***. Il est de ce fait beaucoup plus léger que le protocole DAP qui est un protocole de ***consultation*** et de ***modification*** de l'annuaire sous forme ***binaire***. Les raisons [FORU-95c] suivantes font donc préférer l'utilisation par le poste client du protocole SOLO plutôt que celle du protocole DAP :

- il est beaucoup plus facile de rédiger une requête SOLO qu'une requête DAP;
- une requête SOLO équivalant à plus ou moins quatre requêtes DAP, il est préférable d'utiliser le protocole SOLO sur le réseau public pour en réduire le trafic et les coûts de communication;

- le protocole SOLO repose sur la couche transport TCP alors qu'il faut disposer de toutes les couches du modèle OSI pour utiliser le protocole DAP.

5.3.3 Architecture logique de l'annuaire Forum Lookup

Les modules de base [FORU-95a] nécessaires au fonctionnement de l'annuaire Forum Lookup sont :

- les **IHM** (**interfaces utilisateurs** - PC, Macintosh, Minitel) pour la consultation des informations de l'annuaire;
- le DSA (situé sur la plate-forme Forum Lookup) qui héberge une partie de l'annuaire et en offre l'accès en consultation ainsi qu'en modification;
- les différents serveurs reliant les IHM au DSA via le protocole SOLO :
  - ♦ le **DUA-serveur** qui assure la traduction entre le protocole SOLO et le protocole DAP pour l'interrogation du DSA,
  - ♦ le **SOLO-routeur** qui a pour rôle de faire le relais entre les IHM sur les postes bureautiques en réseau local et le DUA-serveur distant (sur la plate-forme Forum Lookup);
- le **serveur Vidéotex** (sur la plate-forme Forum Lookup) pour l'accès en consultation de l'annuaire à partir d'un Minitel.

Les modules complémentaires que l'on peut installer sur le site du client sont :

- un DSA redondant avec une partie du DSA de la plate-forme Forum Lookup mais dont l'utilité est de diminuer les temps de réponse et de gérer les données localement;
- un **SOLO-serveur** qui transmet les requêtes de consultation provenant des postes utilisateur soit vers l'annuaire de site (il joue le rôle de DUA-serveur), soit vers la plate-forme Forum Lookup (il joue le rôle de SOLO-routeur).

Les différents composants de l'annuaire Forum Lookup sont présentés à la Figure 15.

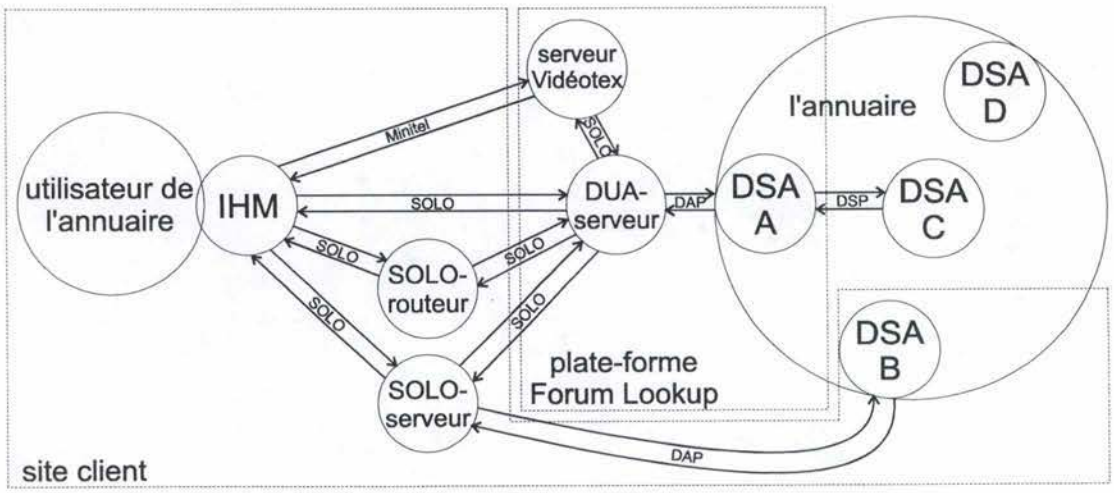


Figure 15 - Architecture logique de Forum Lookup

5.3.4 Architecture physique de l'annuaire Forum Lookup

La plate-forme Forum Lookup [FORU-95a] est composée des machines et des modules suivants :

- deux stations SUN Sparc 20 hébergeant chacune un DUA-serveur et un serveur Vidéotex;
- deux stations SUN Sparc 1000 hébergeant chacune un DSA admettant 75.000 entrées.

Il est important de noter que ces deux DSA gèrent exactement les mêmes entrées. La configuration de cette plate-forme est donc redondante pour des raisons de sécurité et de performance.

L'annuaire Forum Lookup prévoit en outre cinq façons différentes d'accéder à la plate-forme Forum Lookup :

- un Minitel accède au réseau public X.25 en se connectant à un **PAVI** (**Point d'Accès VIdéotex**) via le **RTC** (**Réseau Téléphonique Commuté**);
- un poste isolé accède au réseau public X.25 en se connectant à un **PAD** ("*Packet Assembler Disassembler*") via le RTC grâce à un modem. Ce poste est parfois appelé "poste nomade";
- un poste isolé a un accès direct au réseau public X.25;
- un poste sur **RLE** (**Réseau Local d'Entreprise**) accède à X.25 en passant par un SOLO-routeur;
- un poste sur RLE, disposant d'un annuaire local, a un accès X.25 via un SOLO-serveur.

La Figure 16 illustre les différentes configurations possibles.

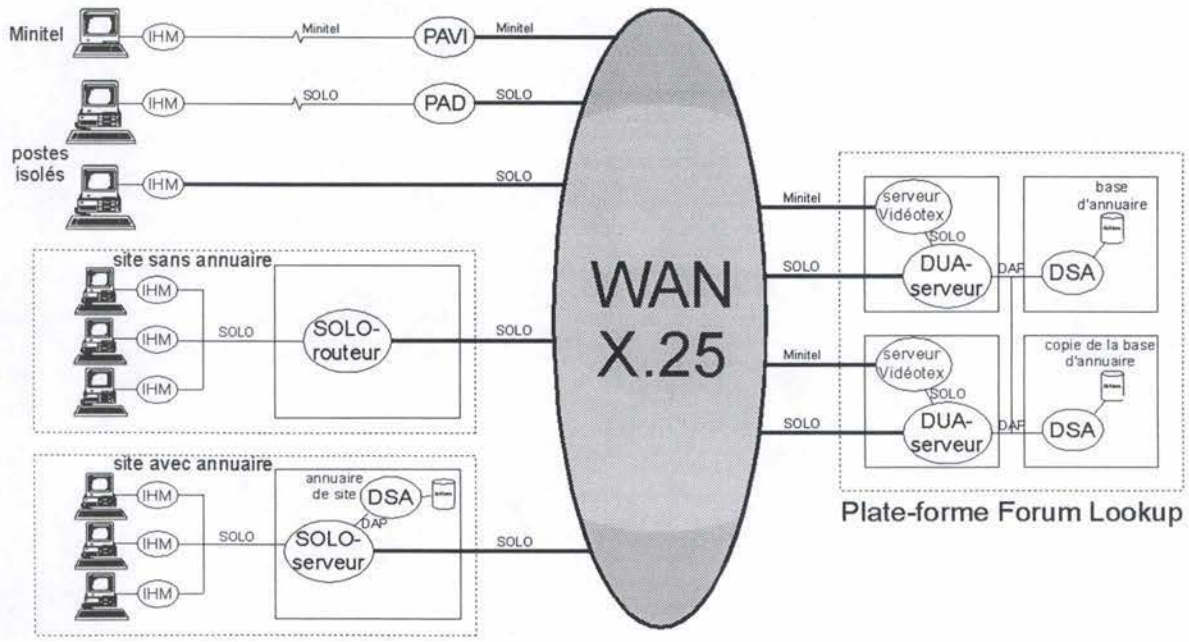


Figure 16 - Architecture physique de Forum Lookup

Expliquons maintenant de façon plus détaillée le rôle des différents composants de l'annuaire Forum Lookup.

5.3.5 DUA-serveur

Le DUA-serveur [FORU-95b] fournit à un utilisateur un accès (par le protocole SOLO) à l'annuaire X.500 en faisant le relais entre, d'une part, les postes isolés, le serveur Vidéotex et les sites ne disposant pas d'un annuaire et, d'autre part, le DSA de la plate-forme Forum Lookup. Son rôle est de :

- résoudre les requêtes de ses clients en traduisant :
    - ♦ les requêtes de consultation utilisant le protocole SOLO en requêtes DAP,
    - ♦ les réponses utilisant le protocole DAP en réponses SOLO;
- basculer dynamiquement sur un DSA de secours si la connexion avec le DSA principal est indisponible.

5.3.6 SOLO-routeur

Sur un RLE ne disposant pas d'un annuaire, le SOLO-routeur [FORU-95c] fournit aux IHM de Forum Lookup un accès à un DUA-serveur distant (par le protocole SOLO). Son rôle consiste à :

- relayer les requêtes venant de plusieurs utilisateurs du RLE sur une connexion SOLO vers le DUA-serveur distant (fonction de centralisation et de multiplexage);
- aiguiller les réponses venant du DUA-serveur vers l'utilisateur approprié (fonction de routage);
- permettre aux IHM de s'affranchir de l'accès direct au **WAN** ("*Wide Area Network*"), limitant ainsi le nombre de connexions ouvertes sur le réseau public.

5.3.7 SOLO-serveur

Sur un RLE disposant d'un annuaire, le rôle du SOLO-serveur [FORU-95d] est de transmettre les requêtes de consultation SOLO (provenant des postes utilisateurs)

- soit vers l'annuaire de site (résolution locale de la requête);
- soit vers l'annuaire situé sur la plate-forme Forum Lookup.

Dans le premier cas, le SOLO-serveur accède au DSA de son site par le protocole DAP (conversion des requêtes SOLO en opérations DAP). Il intègre alors les fonctions de DUA-serveur.

Dans le second cas, le SOLO-serveur envoie la requête au DUA-serveur de la plate-forme Forum Lookup en protocole SOLO sur le réseau public. Il intègre alors les fonctions de SOLO-routeur. Après traitement de la requête, les résultats sont retournés au SOLO-serveur qui les transmet à son client.

L'aiguillage des requêtes se fait en fonction de la répartition des données entre l'annuaire de site et l'annuaire de la plate-forme Forum Lookup.

5.3.8 Serveur Vidéotex

Le serveur Vidéotex [FORU-95a] fait le relais entre les terminaux Minitel et un DUA-serveur. Son rôle est de résoudre les requêtes de ses clients en traduisant :

- les requêtes de consultation Minitel en requêtes SOLO;
- les réponses utilisant le protocole SOLO en réponses Minitel.

5.3.9 IHM

Les IHM [FORU-95a] jouent le même rôle que les DUA que nous avons précédemment présentés (c'est-à-dire fournir à un utilisateur un accès à l'annuaire). Ils ont toutefois des différences au niveau des protocoles : les IHM utilisent le protocole SOLO alors que les DUA utilisent le protocole DAP. Par conséquent, les IHM ne servent qu'à faire des consultations des informations de l'annuaire.

5.4 Compilateur MAVROS

5.4.1 Introduction

A l'heure actuelle, de nombreux réseaux sont composés d'ordinateurs issus de constructeurs différents. Une difficulté apparaissant dans ces réseaux hétérogènes est liée au fait que les ordinateurs n'utilisent pas tous les mêmes formats de donnée, ce qui nécessite des convertisseurs spécifiques pour permettre l'échange d'informations entre ces différents systèmes [ROSE-93b]. Le compilateur MAVROS, développé par Telis (à l'initiative de Christian Huitéma), est un de ces convertisseurs.

MAVROS permet aux programmeurs de spécifier en ASN.1 quelles données seront échangées entre deux applications et de générer automatiquement du code à partir de cette définition. Ceci libère les développeurs d'une grande partie des détails de programmation liés à la gestion et à la transmission des données (y compris la conversion des formats de donnée entre des systèmes hétérogènes) qui peuvent alors se concentrer sur les couches supérieures de l'application.

Dans le cadre de Forum Lookup, MAVROS est entre autre utilisé pour coder et pour décoder les protocoles DAP et DSP ainsi que leurs données.

5.4.2 Code généré

A partir de spécifications écrites en ASN.1, le compilateur MAVROS génère en C des types de donnée (généralement des structures) correspondant aux types de donnée ASN.1 et des routines de codage et de décodage en **BER** ("*Basic Encoding Rules*"), notamment.

En particulier, pour chaque type de données ASN.1 `xxx`, MAVROS génère le type de données C `xxx` et les fonctions suivantes :

```
xxx_cod(...)
```

Cette fonction code une donnée du type `xxx` en syntaxe de transfert BER.

`XXX_dec(...)`

Cette fonction décode une donnée du type XXX à partir de sa représentation en BER.

`XXX_len(...)`

Cette fonction calcule la taille mémoire occupée par le codage en BER d'une donnée du type XXX.

`XXX_free(...)`

Cette fonction libère la mémoire allouée pendant le décodage d'une donnée du type XXX.

`XXX_out(...)`

Cette fonction code une donnée du type XXX en **ASCII** ("*American Standard Code for Information Interchange*"), c'est-à-dire en sa représentation textuelle.

`XXX_in(...)`

Cette fonction décode une donnée du type XXX à partir de sa représentation en ASCII.

`XXX_olen(...)`

Cette fonction calcule la taille mémoire occupée par le codage en ASCII d'une donnée du type XXX.

`XXX_err(...)`

Cette fonction initialise une donnée du type XXX.

`XXX_cpy(...)`

Cette fonction crée une copie d'une donnée du type XXX.

`XXX_cmp(...)`

Cette fonction compare deux données du type XXX.

`XXX_hash(...)`

Cette fonction calcule le code de hachage d'une donnée du type XXX.

5.4.3 Exemple

Pour montrer le travail que MAVROS peut accomplir, nous allons prendre un exemple de spécifications ASN.1 et décrire ce que MAVROS produit à partir de ces spécifications.

Le code qui suit représente la spécification des DN en ASN.1.

```
/* ASN.1 */

DistinguishedName ::=
    SEQUENCE OF
        RelativeDistinguishedName


RelativeDistinguishedName ::=
    SET OF
        AttributeValueAssertion


AttributeValueAssertion ::=
    SEQUENCE {
        type
            AttributeType,

        value
            AttributeValue
    }


AttributeType ::=
    OBJECT IDENTIFIER


AttributeValue ::=
    ANY
```

A partir de la spécification précédente, MAVROS génère les types de donnée suivants (en C) :

```
/* Types de donnée générés */

typedef struct DistinguishedName {
    int n;
    struct RelativeDistinguishedName* v;
} DistinguishedName;


typedef struct RelativeDistinguishedName {
    int n;
    AttributeValueAssertion* v;
} RelativeDistinguishedName;


typedef struct AttributeValueAssertion {
    AttributeType type;
    AttributeValue value;
} AttributeValueAssertion;


typedef asn1_field AttributeType;

typedef asn1_opaque AttributeValue;
```

La structure C, générée par MAVROS, pour représenter un DN est illustrée à la Figure 17.

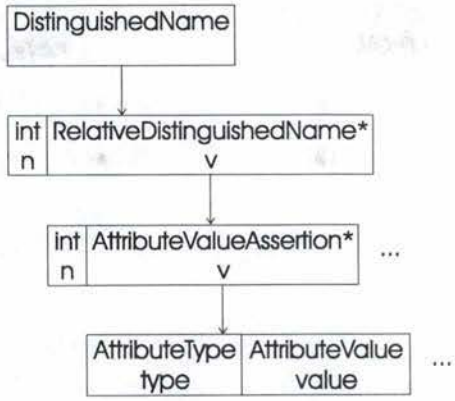


Figure 17 - Structure DistinguishedName

En outre, pour les types de donnée ASN.1 `AttributeValue`, `AttributeType`, `AttributeValueAssertion`, `RelativeDistinguishedName` et `DistinguishedName`, MAVROS produit les fonctions aux suffixes `_cod`, `_dec`, `_len`, `_free`, `_out`, `_in`, `_olen`, `_err`, `_cpy`, `_cmp` et `_hash`.

5.5 Couches de communication

5.5.1 Introduction

La communauté Internet utilise un ensemble de protocoles de transport et de réseau bien développés et largement diffusés : il s'agit des protocoles TCP et IP. D'autre part, les couches session, présentation et application du monde OSI ont aussi été adoptées par la communauté internationale et sont proposées par de nombreux vendeurs. Dans un tel contexte, on trouve désirable de pouvoir offrir ces couches de haut niveau sur les services Internet existants [RFC-1006].

Attendu que les deux modèles proposés par le monde Internet et le monde OSI sont des modèles en couches et que le principe fondamental des modèles en couches est l'indépendance des différentes couches, il devient envisageable de "porter" une application OSI dans un environnement TCP/IP.

Rappelons que le concept d'indépendance des couches exprime le fait que l'utilisateur d'un service est indifférent du protocole utilisé par les entités qui lui offrent ce service tant que le service qui lui est offert est préservé.

Nous allons maintenant décrire une méthode permettant d'implémenter des services de transport OSI sur un réseau TCP/IP afin de mettre en évidence le rôle d'une couche que nous appellerons "couche RFC-1006". Ensuite, nous parlerons des "couches de communication" développées à Telis et de leur intégration dans Forum Lookup.

5.5.2 Service de transport OSI sur TCP/IP

Ce point traite de l'implémentation de services de transport OSI sur des systèmes TCP/IP. En d'autres termes, il décrit une façon de s'y prendre pour définir une couche transport qui offre les mêmes services et les mêmes interfaces que la couche transport OSI alors qu'elle est implémentée sur un protocole réseau de type TCP/IP.

La méthode préconisée par le [RFC-1006] pour atteindre cet objectif consiste à implémenter le protocole TP0 sur TCP/IP. Notons que, puisque TP0 sera implémenté sur TCP/IP, il offrira un service du même niveau de fiabilité que TP4.

Sans entrer dans des détails techniques que nous jugeons inutiles pour notre exposé, nous pouvons toutefois tenter de résumer les principes auxquels doit se soumettre cette implémentation en question de la couche TP0 (que nous appellerons à l'avenir "couche RFC-1006").

La couche RFC-1006 doit offrir à sa couche supérieure (la couche session, en l'occurrence) un service de transport OSI. Pour offrir ce service, elle doit utiliser les services de sa couche inférieure (la couche TCP, en l'occurrence).

La seule difficulté apparente réside justement dans le fait que la couche RFC-1006 est implémentée sur une couche TCP et non sur une couche réseau OSI. Le remède consiste donc à émuler les services réseau OSI dont a besoin la couche RFC-1006 (pour fournir ses services) à partir des services offerts par TCP. Cette correspondance assez immédiate entre primitives de services est complètement détaillée dans le [RFC-1006] (pour le lecteur intéressé).

Grâce à cette correspondance entre primitives de service, la couche RFC-1006 peut offrir à la couche session un service tout à fait identique à celui qu'elle offrirait si elle était implémentée sur la couche réseau OSI. L'indépendance des couches est donc bien garantie.

5.5.3 Philosophie des couches de communication

Une des équipes de Telis a développé des couches de communication utiles aux développeurs d'applications de haut niveau.

Les **couches de communication** sont composées d'une couche transport, d'une couche session, d'une couche présentation et des éléments de service ACSE et ROSE. Elles uniformisent l'utilisation des services de communication, quels que soient les environnements sous-jacents (TCP/IP, OSI, ...).

Du point de vue de l'utilisateur, elles fournissent une interface de communication identique, quelle que soit la machine et quel que soit le type de transport (donc, le réseau).

Pour fournir une indépendance vis-à-vis du type de transport, les couches de communication offrent une interface transport propriétaire qui uniformise l'utilisation des services de transport.

Plus précisément, la couche transport des couches de communication est une couche homogène qui offre des services transport génériques tout à fait indépendants des véritables services de transport utilisés. En d'autres termes, elle émule des primitives de service générales à partir des primitives de service particulières qui sont réellement à sa disposition (et qui, elles, dépendent de la configuration propre de chaque machine).

5.5.4 Services offerts

Les couches de communication permettent de rendre le fonctionnement des applications de haut niveau totalement indépendant des moyens réellement utilisés lorsqu'elles communiquent.

Les services de communication qu'elles offrent sont : l'établissement d'une connexion, l'envoi de données, la réception de données et la fermeture d'une connexion.

5.5.5 Couches inférieures

A l'heure actuelle, les couches de communication de Telis sont capables d'utiliser les services fournis par les quatre types d'environnement réseau suivants :

- TCP/IP;
- RFC-1006/TCP/IP;
- TP0/X.25;
- XModem/RTC.

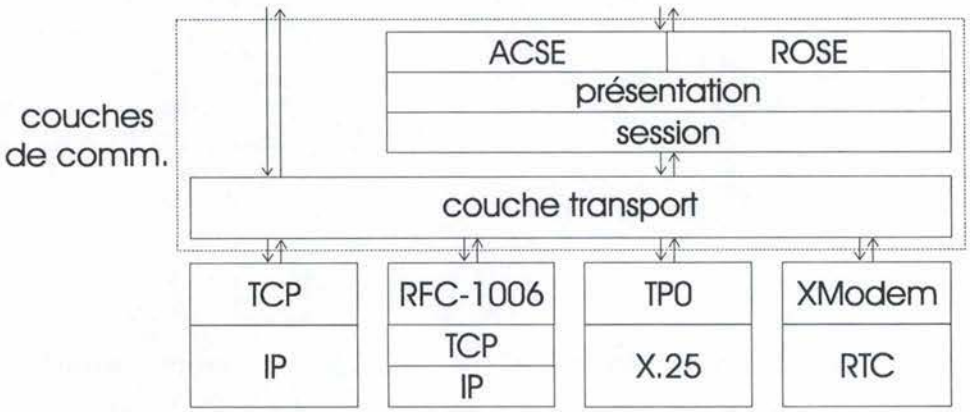


Figure 18 - Couches de communication de Telis

La Figure 18 illustre les couches de communication de Telis et les couches qui leur sont inférieures.

L'interface transport TCP/IP n'est jamais utilisée avec les couches supérieures des couches de communication. C'est un cas particulier qui a été développé pour fournir un accès Internet (transport) aux serveurs SOLO.

Notons que, si les besoins s'en font sentir, rien n'empêche les couches de communication de pouvoir gérer de nouveaux environnements réseau (UDP/IP, par exemple). Ceci est vrai grâce au principe d'indépendance des différentes couches.

5.5.6 Choix d'un service de transport particulier

Si l'on désire établir une connexion avec une entité application, on doit passer aux couches de communication le **point d'accès** (PSAP) de cette entité. C'est par le biais de cette adresse que l'on indique le véritable service de transport sous-jacent que l'on veut utiliser comme moyen de communication.

Par exemple, la PSAP "`Internet-RFC-1006=uranus+2345`" localise une entité à laquelle on veut accéder par un transport RFC-1006 sur un réseau TCP/IP. Cette entité réside dans la machine "`uranus`" au port `2345`.

D'autres préfixes sont utilisés pour désigner les autres types de transport (par exemple, "`localPureTCP`" désigne l'interface transport TCP/IP).

5.5.7 Utilisation dans Forum Lookup

Maintenant que nous avons détaillé le rôle des couches de communication, nous pouvons nous attarder quelque peu sur l'utilisation qui en est faite dans le cadre de Forum Lookup.

La Figure 19 propose une vue détaillée de tous les protocoles sous-jacents à l'IHM, au SOLO-routeur, au DUA-serveur et au DSA. Elle est à mettre en rapport avec l'architecture logique de l'annuaire Forum Lookup que nous avions présentée à la Figure 15.

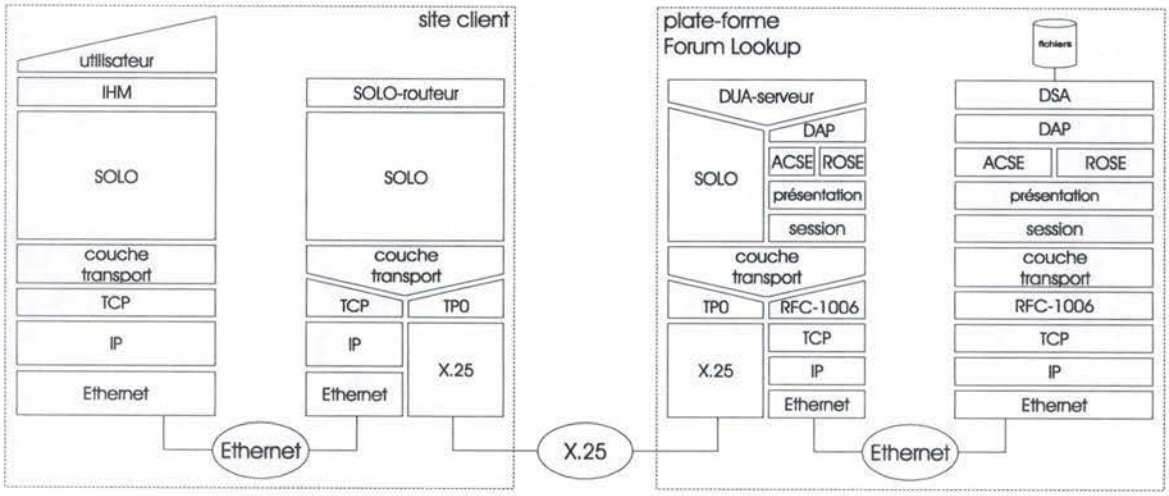


Figure 19 - Architecture logique et couches de communication

Bien que nous ne décrirons pas chacun des détails constituant cette figure, nous allons attirer l'attention du lecteur sur les éléments les plus importants qu'il ne connaît pas encore.

La première constatation à faire est que les couches de communication se retrouvent sur toutes les machines (du site client à la plate-forme Forum Lookup).

Le deuxième constatation est que l'implémentation du protocole SOLO, faite à Telis, se base sur les services transport des couches de communication et non pas sur les services transport TCP (comme cela est prévu dans [HUIT-94]). Cette implémentation du protocole SOLO est donc plus "portable" qu'elle l'aurait été en appliquant strictement les recommandations du "*draft*" SOLO. Ainsi, les requêtes SOLO peuvent être véhiculées dans des réseaux ne supportant pas le protocole TCP/IP.

La troisième constatation (corollaire de la deuxième) est que, justement, SOLO s'appuie indirectement sur la couche transport TP0 à travers le réseau public X.25 (entre le SOLO-routeur et le DUA-serveur). SOLO utilise cependant un service transport strictement équivalent au service transport TCP entre l'IHM et le SOLO-routeur.

Finalement, constatons que le DSA utilise indirectement les couches TCP/IP pour communiquer avec le DUA-serveur.

6. SOLO

6.1 Introduction

6.1.1 But du service

Le protocole SOLO, imaginé par Christian Huitéma [HUIT-94], est un protocole simplifié d'accès à un service d'annuaire distribué à partir de postes bureautiques. Il est basé sur l'expérience acquise dans plusieurs implémentations de X.500 et permet d'obtenir des gains de performance par rapport au DAP.

6.1.2 Motivations

Voici différentes raisons [FORU-95h] qui sont en faveur de SOLO comme protocole d'accès au service d'annuaire :

- X.500 est beaucoup trop complexe pour la plupart des services d'annuaire (modèle hiérarchique, opérations complexes), ceux-ci n'utilisant qu'une partie des fonctionnalités X.500;
- `whois++` est trop différent de X.500 (modèle de données, noms);
- il faut pouvoir intégrer les services d'annuaire avec le **WWW** ("*World-Wide Web*").

6.1.3 Principes

Voici les principes généraux [FORU-95g] du protocole SOLO :

- SOLO est indépendant de la technologie sous-jacente du côté du serveur : X.500, `whois`, **SQL** ("*Structured Query Language*"), ...;
- il utilise TCP/IP ou UDP/IP (dans l'implémentation suggérée par le "*draft*" [HUIT-94]);
- les requêtes et les réponses SOLO sont codées en format texte (ASCII);
- SOLO emprunte à X.500 le modèle des données et le modèle de nommage (X.500 simplifié);
- il emprunte à `whois++` le modèle de navigation entre serveurs (concept de "*centroïd*" auquel nous nous intéresserons plus tard);
- il admet la syntaxe UFN dans les requêtes et les réponses;
- le langage de requêtes est simple;
- SOLO s'intègre bien avec d'autres services d'information comme le "*Web*" : on peut envisager que les requêtes SOLO soient des **URL** ("*Universal Resource Locator*") et que les réponses correspondant à ces requêtes soient des pages formatées en **HTML** ("*HyperText Markup Language*").

Bien que largement inspiré par le protocole X.500, SOLO n'essaie pas de suivre le modèle X.500. Néanmoins, le service qu'il offre est suffisamment puissant pour que la réalisation d'une passerelle entre les protocoles d'accès X.500 et SOLO soit aisée.

6.2 Services offerts par l'entité SOLO

Nous allons présenter les différentes commandes SOLO (illustrées à la Figure 20) en les accompagnant chaque fois par quelques exemples.

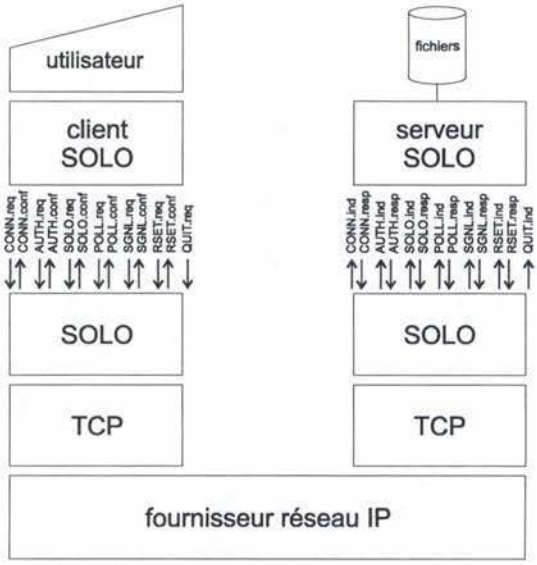


Figure 20 - Architecture et primitives de service SOLO

6.2.1 Connexion

La primitive *CONN* permet à un client d'établir une connexion TCP/IP avec un serveur SOLO.

Paramètre de *CONN.req* :

- nom d'un serveur.

```
CONN.req(uranus.telis-sc.fr:2222)
```

Paramètre de *CONN.conf* :

aucun ...

```
CONN.conf()
```

6.2.2 Authentification

La primitive *AUTH* permet à un utilisateur de s'authentifier (ce qui permet d'identifier ses droits).

Cette requête d'authentification d'un utilisateur est propre à Forum Lookup. Elle a été ajoutée au protocole SOLO tel qu'il est présenté dans [HUIT-94].

Paramètres de *AUTH.req* :

- identifiant de la requête; .

- politique d'authentification (soit "simple", soit "none");
- données d'authentification (dans le cas où la politique d'authentification est "simple") :
    - nom (UFN) d'un utilisateur,
    - mot de passe de cet utilisateur.

```
AUTH.req(12345, simple, <interne,e3x,fr>, interne)
```

Paramètres de *AUTH.conf* :

- identifiant de la requête;
- message d'information.

```
AUTH.conf(12345,
        "600 Ready.")
```

6.2.3 Consultation

La primitive *SOLO* permet de consulter l'information associée à une entrée.

Paramètres de *SOLO.req* :

- identifiant de la requête;
- compteur de relais (optionnel; valeur par défaut : 0); (voir plus loin)
- nom (UFN) d'une entrée;
- spécificateur de précision (soit "?", soit "!"); (voir ci-dessous)
- liste d'attributs (optionnelle; valeur par défaut : tous les attributs).

Paramètres de *SOLO.conf* :

- identifiant de la requête;
- message(s) d'information (un ou plusieurs);
- liste de valeurs d'attribut (optionnelle).

Nous allons maintenant examiner différents cas possibles de consultation sur base d'une liste non exhaustive d'exemples :

- nom exact

  Dans ce cas, l'utilisateur veut spécifier complètement le nom de l'entrée demandée, en ne laissant aucune possibilité d'interprétation au serveur. Cela est spécifié dans la requête du dialogue suivant par le point d'exclamation ("!") entre le nom de l'entrée et la liste des attributs souhaités :

  ```
  SOLO.req(12346, <CN=Ascan Woermann,OU=osi,O=e3x,C=fr>, !, [Phone, Email])

  SOLO.conf(12346,
          "500 Matches: <CN=Ascan Woermann,OU=osi,O=e3x,C=fr>",
          [Phone: +33 93.95.40.69, Email: woermann@sophia.telis-sc.fr])
  ```

  Une requête de ce type est totalement équivalente, au niveau du service offert, à l'opération *Read* en DAP.

- recherche synonymique
  En indiquant un point d'interrogation ("?") entre le nom de l'entrée demandée et la liste des attributs souhaités, l'utilisateur spécifie que le nom de l'entrée n'est pas nécessairement un nom exact.
  Ainsi, dans l'exemple ci-dessous, l'entrée trouvée par le serveur contient un attribut dont une des valeurs non spécifiques est la référence "A. Woermann" recherchée par le client.

```
SOLO.req(12348, <A. Woermann, osi, e3x, fr>, ?, [Phone, Email])

SOLO.conf(12348,
        "500 Matches: <CN=Ascan Woermann,OU=osi,O=e3x,C=fr>",
        [Phone: +33 93.95.40.69, Email: woermann@sophia.telis-sc.fr])
```

- recherche ambiguë
  Dans la requête suivante, la référence "dupont" est ambiguë. En effet, il y a plusieurs "dupont" dans l'équipe "osi". Le serveur fournit alors la liste de tous les candidats qui satisfont la requête.

```
SOLO.req(12347, <dupont, osi, e3x, fr>, ?, [Phone, Email])

SOLO.conf(12347,
        "201-Ambiguous name: <dupont, osi, e3x, fr>",
        "301-Partial Match: <osi, e3x, fr> <OU=osi,O=e3x,C=fr>",
        "400-Suggestion: <CN=Laurence Dupont,OU=osi,O=e3x,C=fr>",
        "400 Suggestion: <CN=Yves Dupont,OU=osi,O=e3x,C=fr>")
```

- recherche approximative (phonétique)
  Dans le dialogue qui suit, la référence "worman" est incorrecte. Le serveur applique alors une stratégie locale pour essayer de déduire des références valides. Ici, il essaie une égalité "phonétique" sur la référence erronée.

```
SOLO.req(12350, <worman, osi, e3x, fr>, ?, [Phone, Email])

SOLO.conf(12350,
        "202-No such name: <worman, osi, e3x, fr>",
        "301-Partial Match: <osi, e3x, fr> <OU=osi,O=e3x,C=fr>",
        "400 Suggestion: <CN=Ascan Woermann,OU=osi,O=e3x,C=fr>")
```

- recherche par nom tronqué (sous-chaîne)
  La recherche suivante est une recherche par sous-chaîne. Le serveur suggère toutes les entrées contenant la sous-chaîne "woer".

```
SOLO.req(12349, <woer*, osi, e3x, fr>, ?, [Phone, Email])

SOLO.conf(12349,
        "202-No such name: <woer*, osi, e3x, fr>",
        "301-Partial Match: <osi, e3x, fr> <OU=osi,O=e3x,C=fr>",
        "400 Suggestion: <CN=Ascan Woermann,OU=osi,O=e3x,C=fr>")
```

  Notons que l'étoile ("*") est facultative, le serveur passant automatiquement en recherche par sous-chaîne s'il ne trouve rien en recherche exacte et en recherche phonétique.

- recherche redirigée
  Il arrive qu'un serveur SOLO ne puisse pas répondre à une requête. Dans ce cas, il peut fournir une indication au client sur un "serveur suggéré" où il pourra soumettre sa requête (renvoi d'une référence) :

```
SOLO.req(12351, <woermann, osi, e3x, fr>, ?, [Phone, Email])

SOLO.conf(12351,
        "301-Partial Match: <e3x, fr> <O=e3x,C=fr>",
        "302-Suggested-server: <O=e3x,C=fr> sophia.telis-sc.fr",
        "302-Suggested-server: <O=e3x,C=fr> mitsou.inria.fr",
        "400 Suggestion: <woermann, osi,O=e3x,C=fr>")
```

Remarquons que le serveur aurait aussi pu décider de relayer lui-même la requête vers un ou plusieurs autres serveurs (requête chaînée). Dans ce cas, il expédie la requête aux autres serveurs en utilisant le protocole SOLO et, une fois qu'il a obtenu le résultat, il le renvoie au client.

Le protocole SOLO laisse beaucoup de latitude au serveur qui décide de relayer une requête (choix du nombre de serveurs à tester, façon de les tester, ...). Mais le protocole impose tout de même qu'une indication de la décision prise (en particulier, une indication de la provenance des données) soit retournée au client.

Dans l'exemple qui suit, le serveur suggère deux serveurs ("`mitsou.inria.fr`" et "`sophia.telis-sc.fr`") et décide de relayer la requête de consultation de l'entrée `<woermann, osi,O=e3x,C=fr>` vers le serveur "`sophia.telis-sc.fr`".

```
SOLO.req(12352, <woermann, osi, e3x, fr>, ?, [Phone, Email])

SOLO.conf(12352,
        "301-Partial Match: <e3x, fr> <O=e3x,C=fr>",
        "302-Suggested-server: <O=e3x,C=fr> sophia.telis-sc.fr",
        "302-Suggested-server: <O=e3x,C=fr> mitsou.inria.fr",
        "402-Relaying: <woermann, osi,O=e3x,C=fr> sophia.telis-sc.fr",
        "500 Matches: <CN=Ascan Woermann,OU=osi,O=e3x,C=fr>",
        [Phone: +33 93.95.40.69, Email: woermann@sophia.telis-sc.fr])
```

Puisque les requêtes peuvent être relayées, le protocole a inclus une protection contre le bouclage via l'insertion d'un "compteur de relais" (optionnel) dans la requête SOLO. Dans l'exemple qui suit, on insère, entre l'identifiant de la requête et le nom de l'entrée, un compteur de relais initialisé à 1 :

```
SOLO.req(12353, 1, <woermann, osi, e3x, fr>, ?, [Phone, Email])
```

Les serveurs SOLO doivent incrémenter le compteur de relais lorsqu'ils expédient la requête à un autre serveur. Toute requête dont le compteur de relais est supérieur ou égal à 8 ne peut pas être relayée. Un compteur de relais absent est équivalent à un compteur égal à 0.

- recherche d'un pointeur sur l'information
  Il est parfois peu pratique pour un serveur de fournir complètement la valeur d'un attribut : certains objets comme des photographies ou des bandes sonores sont trop volumineux pour être facilement transmis. D'autres peuvent ne pas être disponibles sur le serveur SOLO lui-même. Dans ces cas, le serveur peut décider de retourner un pointeur sur la valeur sous la forme d'une URL à la place de la valeur elle-même :

```
SOLO.req(12354, <woermann, osi, e3x, fr>, ?, [Photo, Email])

SOLO.conf(12354,
        "500 Matches: <CN=Ascan Woermann,OU=osi,O=e3x,C=fr>",
        [-Photo:"http://toscana.telis-sc.fr:2220/people/woermann/photo",
         Email: woermann@sophia.telis-sc.fr])
```

La présence d'une URL à la place de la valeur est signalée par un trait d'union devant le type d'attribut.

Certains clients peuvent même vouloir forcer une réponse de ce type, pour obtenir la localisation de la valeur d'un attribut sans devoir transférer la donnée. Les clients peuvent réaliser cela en préfixant le type d'attribut considéré par un trait d'union dans leur requête :

```
SOLO.req(12355, <woermann, osi, e3x, fr>, ?, [-Photo, Email])

SOLO.conf(12355,
          "500 Matches: <CN=Ascan Woermann,OU=osi,O=e3x,C=fr>",
          [-Photo:"http://toscana.telis-sc.fr:2220/people/woermann/photo",
           Email: woermann@sophia.telis-sc.fr])
```

6.2.4 Importation de la connaissance

La primitive *POLL* permet à un client ou à un serveur d'obtenir la **connaissance** d'un autre serveur (c'est-à-dire l'ensemble des attributs indexés et, pour chacun de ceux-ci, la liste des valeurs trouvées dans n'importe quelle entrée de la base de données). Toute cette connaissance est stockée dans une très grande entrée appelée "*centroïd*".

Cette requête n'est pas utilisée dans Forum Lookup.

Paramètres de *POLL.req* :

- identifiant de la requête;
- nom d'un type ("*template*") (optionnel; valeur par défaut : tous les types, c'est-à-dire "`personne`", "`domaine`" et "`document`");
- liste d'attributs (optionnelle).

Paramètres de *POLL.conf* :

- identifiant de la requête;
- message d'information;
- données :
  - ♦ liste de valeurs d'attribut (dans le cas où la requête comportait une liste d'attributs),
  - ♦ liste d'attributs (dans le cas contraire).

Le client peut obtenir la liste des attributs maniés par le serveur en envoyant la requête suivante :

```
POLL.req(12356)
```

Le serveur renvoie alors la liste des attributs qu'il est prêt à exporter :

```
POLL.conf(12356,
          "502 Providing attribute list.",
          [S, OU, O, C, Template])
```

Si le client désire maintenant obtenir les valeurs associées à ces attributs, il lui reste à envoyer la commande suivante :

```
POLL.req(12357, [S, OU, O, C, Template])
```

Le serveur renverra alors une réponse semblable à celle-ci :

```
POLL.conf(12357,
        "501 Sending indexes.",
        [S:*, OU:OSI,R&D, O:E3X, C:FR, Template:person,domain])
```

On remarquera que le serveur ne fournit pas la liste des valeurs associées à l'attribut **S** ("*Surname*") mais la remplace par une étoile ("*"). Cette pratique est utilisée pour des raisons de sécurité. Il peut être en effet très dangereux de fournir la liste de tous les employés d'une entreprise. De plus, dans le cas où la liste est longue, cette pratique se justifie alors aussi pour des raisons de commodité.

6.2.5 Notification de la présence d'un serveur

La primitive *SGNL* permet à l'administrateur d'un serveur de lui signaler la présence d'un nouveau serveur ou la mise à jour d'une base de données distante.

Cette requête n'est pas utilisée dans Forum Lookup.

Paramètres de *SGNL.req* :

- identifiant de la requête;
- nom d'un serveur.

```
SGNL.req(12358, mitsou.inria.fr:8990)
```

Paramètres de *SGNL.conf* :

- identifiant de la requête;
- message d'information.

```
SGNL.conf(12358,
        "601 Will poll: mitsou.inria.fr:8990")
```

Dans cette réponse, le serveur annonce qu'il a prévu d'émettre la commande *POLL* vers le serveur signalé.

Comme on le voit, les primitives *SGNL* et *POLL* permettent de mettre à jour les informations de navigation dans un réseau de serveurs SOLO. En effet, *SGNL* permet de notifier qu'un nouveau serveur est opérationnel ou qu'une base de données distante a été modifiée et *POLL* permet d'importer la connaissance de ce serveur distant.

6.2.6 Abandon de requêtes

La primitive *RSET* sert à annuler une ou plusieurs requêtes de consultation en cours.

Paramètres de *RSET.req* :

- identifiant de la requête;
- identifiant d'une requête (optionnel; valeur par défaut : les identifiants de toutes les requêtes en cours).

Paramètres de *RSET.conf* :

- identifiant de la requête;
- message(s) d'information (un ou plusieurs).

Supposons que le client ait lancé les requêtes suivantes :

```
SOLO.req(12359, <heuse, osi, e3x, fr>, ?, [Phone])
SOLO.req(12360, <huitema, inria, fr>, ?, [Phone])
```

Si le client désire arrêter d'attendre les réponses à ces requêtes en cours de résolution, il peut émettre la requête suivante qui aura pour effet de les annuler toutes d'un coup :

```
RSET.req(12361)

SOLO.conf(12359,
        "108 Abandoning the request.")
SOLO.conf(12360,
        "108 Abandoning the request.")
RSET.conf(12361,
        "600 Ready.")
```

Il aurait aussi bien pu annuler les requêtes séparément :

```
RSET.req(12362, 12359)

SOLO.conf(12359,
        "108 Abandoning the request.")
RSET.conf(12362,
        "600 Ready.")


RSET.req(12363, 12360)

SOLO.conf(12360,
        "108 Abandoning the request.")
RSET.conf(12363,
        "600 Ready.")
```

6.2.7 Déconnexion

La primitive (non confirmée) *QUIT* permet de fermer la connexion établie avec le serveur après l'arrêt des requêtes en cours.

Paramètre de *QUIT.req* :

- identifiant de la requête.

```
QUIT.req(12364)
```

6.3 Protocole

6.3.1 Couches inférieures

Les opérations SOLO passent sur des connexions TCP/IP ou dans des datagrammes UDP (quand les performances représentent un critère important). Dans ce dernier cas, toutes les commandes SOLO sont supportées (sauf les commandes *CONN*, *AUTH*, *RSET* et *QUIT* qui n'ont pas de signification dans ce contexte) sous la contrainte de la taille d'un datagramme.

Toute requête et toute réponse doivent être envoyées dans un seul datagramme. Dès lors, pour rendre les réponses SOLO plus compactes, le texte libre associé aux messages d'informations dans les réponses ou les messages d'erreur est souvent omis, comme le montre l'exemple suivant :

```
SOLO.req(12350, <worman, osi, e3x, fr>, ?, [Phone, Email])
```

```
SOLO.conf(12350,
        "202-: <worman, osi, e3x, fr>",
        "301-: <osi, e3x, fr> <OU=osi,O=e3x,C=fr>",
        "400 : <CN=Ascan Woermann,OU=osi,O=e3x,C=fr>")
```

6.3.2 Liste des PDU

6.3.2.1 Format des requêtes

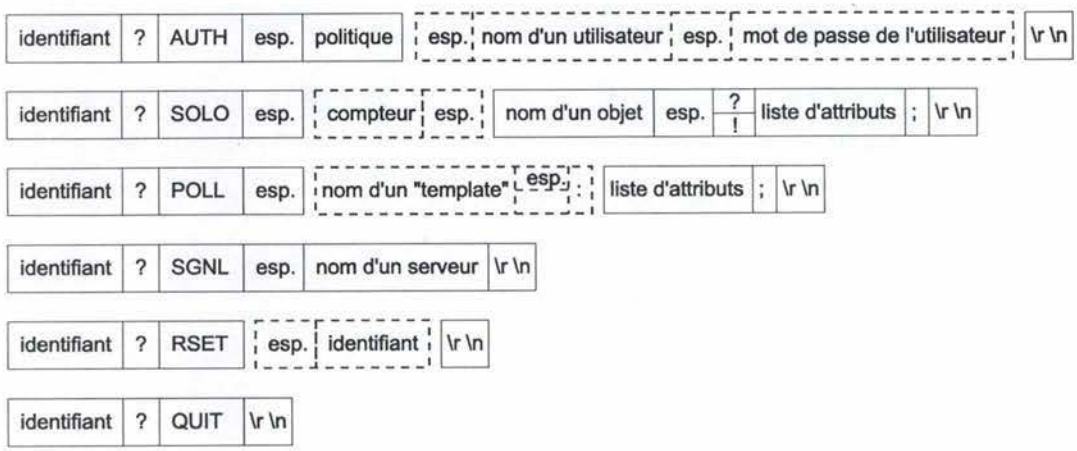


Figure 21 - Format des PDU de requête SOLO

La Figure 21 présente les formats des PDU de requête SOLO [HUIT-94]. Les PDU, codés en ASCII, commencent tous par :

1) un identifiant, encodé sur 5 chiffres décimaux, choisi par le client pour être unique dans le contexte de la connexion;
2) le caractère "?";
3) un code de requête sur 4 lettres : "AUTH", "SOLO", "POLL", "SGNL", "RSET" ou "QUIT" (en minuscules, majuscules ou un mélange des deux).

La requête "AUTH" inclut en plus [FORU-95b] :

1) un nombre variable de caractères d'espacement (espaces ou tabulations);
2) une politique d'authentification qui peut être soit "simple", soit "none";
3) le nom d'un utilisateur (précédé et suivi par un nombre variable de caractères d'espacement) et le mot de passe associé à cet utilisateur, dans le cas où la politique d'authentification est "simple".

La requête "SOLO" inclut en plus :

1) un nombre variable de caractères d'espacement;
2) un "compteur de relais" optionnel, codé sur un chiffre décimal, suivi, s'il est présent, par un nombre variable de caractères d'espacement;

3) le nom de l'entrée demandée;

4) un nombre variable de caractères d'espacement;

5) un spécificateur de précision qui peut être soit un point d'interrogation ("?"), soit un point d'exclamation ("!");

6) une liste d'attributs, terminée par un point-virgule (";").

La requête "POLL" inclut en plus :

1) un nombre variable de caractères d'espacement;

2) un nom de "*template*" optionnel, suivi s'il est présent, par un nombre variable (y compris 0) de caractères d'espacement et par les deux-points (":");

3) une liste d'attributs, terminée par un point-virgule (";").

La requête "SGNL" inclut en plus :

1) un nombre variable de caractères d'espacement;

2) le nom du serveur dont on veut importer les connaissances.

La requête "RSET" peut, éventuellement, être suivie d'un nombre variable de caractères d'espacement et d'un identifiant de requête codé sur 5 chiffres décimaux.

La requête "QUIT", elle, ne contient aucune information supplémentaire.

Toutes les requêtes sont terminées par un indicateur de "fin de ligne", composé des deux caractères "retour chariot" et "nouvelle ligne".

6.3.2.2 Format des réponses

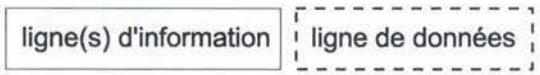


Figure 22 - Format des PDU de réponse SOLO

Les réponses SOLO, qui font l'objet de la Figure 22, sont toujours composées d'une ou de plusieurs lignes d'information et parfois suivies d'une ligne de données.

Figure 23 - Format des lignes d'information

Les lignes d'information, présentées à la Figure 23, sont toujours composées de :

1) l'identifiant fourni dans la requête correspondante, codé sur 5 chiffres décimaux;

2) le caractère deux-points (":");

3) un code d'information sur 3 chiffres;

4) un caractère de continuation, c'est-à-dire un espace si c'est la dernière ligne d'information ou un trait d'union ("-") sinon;

5) un texte libre composé de lettres, d'espaces, de chiffres, de tirets et de virgules.

Si la ligne ne contient pas d'information supplémentaire, le texte libre est terminé par un point (".") . Sinon, le texte libre est suivi par les deux-points (":") et l'information. Cette information supplémentaire est composée :

- soit du nom d'une entrée,
- soit du nom d'une entrée, suivi du nom d'une autre entrée,
- soit du nom de domaine d'un serveur,
- soit d'une URL,
- soit du nom d'un serveur.

Le texte libre est optionnel. Néanmoins, le point terminal (".") ou les deux-points (":") doivent être présents. Dans le cas des deux-points, l'information supplémentaire doit être fournie.

Toutes les lignes d'information sont terminées par un indicateur de "fin de ligne".

liste d'attributs liste de valeurs d'attributs	\r \n

Figure 24 - Format des lignes de données

Quand le premier chiffre de la dernière ligne d'information a la valeur "5", cette ligne est suivie par une liste d'attributs (*POLL*) ou par une liste de valeurs d'attribut (*SOLO* et *POLL*). Et, bien que cela ne soit pas précisé dans le "*draft*" [HUIT-94], nous supposons que les lignes de données (illustrées à la Figure 24) se terminent aussi par un indicateur de "fin de ligne".

6.3.2.3 Format des composants des requêtes et des réponses

Pour être complet, il nous reste maintenant à examiner le format de différents composants des requêtes et des réponses :

- format des spécifications de nom
  Les noms sont des DN sous forme de chaînes de caractères [RFC-1485] ou des représentations textuelles de noms formatés suivant la syntaxe UFN [RFC-1484]. Toutefois, SOLO admet certaines extensions et impose d'autres restrictions par rapport à ces deux spécifications :
  - ♦ l'utilisation du seul caractère "*" comme spécification d'un des composants d'un nom (recherche par sous-chaîne) est interdite;

- l'utilisation du caractère "|" comme opérateur disjonctif est autorisée. Dès lors, ce caractère doit être mis entre guillemets s'il est présent dans une valeur d'attribut (pour pouvoir le distinguer de l'opérateur);
- la restriction imposée par la spécification UFN sur l'utilisation du délimiteur "+" ne s'applique pas en SOLO : plusieurs assertions peuvent se trouver dans un composant du nom sans pour autant être explicites. On peut en effet omettre de spécifier les types des attributs.

- format des noms de serveur
  Les noms des serveurs sont composés d'un nom de domaine éventuellement suivi par un deux-points et un numéro de port. Si le numéro de port est omis, la valeur par défaut est la valeur <solo>.

- format des listes d'attributs
  Les listes d'attributs sont composées de noms d'attribut séparés par des virgules. Un nombre arbitraire de caractères d'espacement peut être inséré autour des noms d'attribut. Notons qu'une liste d'attributs peut être vide.

  Dans les requêtes, les listes d'attributs sont toujours terminées par un point-virgule. Dans les réponses, par contre, elles sont toujours terminées par une ligne contenant seulement un point.

  Dans la commande "SOLO", les types d'attribut peuvent être précédés par :

  - un trait d'union ("-"), pour indiquer le désir de recevoir des pointeurs sur les valeurs (URL) plutôt que les valeurs elles-mêmes;
  - un plus ("+"), pour obtenir des informations (de type administratif) dont nous choisissons de ne pas parler ici.

- format des listes de valeurs d'attribut
  Une liste de valeurs d'attribut est composée de plusieurs valeurs d'attribut, chacune d'entre elles étant encodée sur une ou plusieurs lignes de texte. Toutes les lignes de texte sont terminées par un indicateur de "fin de ligne". La liste est terminée par une ligne contenant un simple point. La liste peut être vide, c'est-à-dire qu'elle peut contenir pour seule ligne la ligne contenant un point.

  La première ligne d'une valeur d'attribut commence par l'encodage du type d'attribut, suivi par un deux-points. Toutes les autres lignes devraient commencer par un ou plusieurs caractères d'espacement. Chaque ligne contient une ou plusieurs valeurs séparées de la valeur suivante par une virgule et un nombre arbitraire d'espaces.

  Dans les valeurs retournées par la commande "SOLO", le type d'attribut peut éventuellement être précédé d'un trait d'union. Dans ce cas, les valeurs retournées sont des URL, au lieu des véritables valeurs.

  Toujours dans le cas de la commande "SOLO", le type d'attribut peut être précédé par un "+" mais nous choisissons de ne pas développer ce cas.

- format des noms d'attribut
  Les noms d'attribut peuvent être de simple mots clés ou des traductions en format texte d' "*object identifiers*".
- format des valeurs
  Une valeur peut être :
  - un ensemble de caractères imprimables, excepté les caractères spéciaux suivants : la virgule (","), les deux-points (":"), l'égal ("="), le point-virgule (";"), le point d'interrogation ("?"), le caractère inférieur ("<") et le caractère supérieur (">"),
  - une chaîne de caractères entre guillemets,
  - une chaîne de bits entre apostrophes.
- format des "*templates*"
  Les noms d'attribut peuvent être de simples mots clés ou des traductions en format texte d' "*object identifiers*" de "*templates*".

6.3.3 Exemples

Nous allons maintenant brièvement illustrer par des exemples les PDU échangés entre un serveur SOLO et un client.

Si un serveur SOLO se trouve sur le port `2222` de la machine "`uranus.telis-sc.fr`", l'utilisateur (agissant en tant que client SOLO) peut établir une connexion avec lui en introduisant la commande suivante :

```
telnet uranus.telis-sc.fr 2222
```

Cette commande est, en effet, la forme concrète de la primitive de service à invoquer pour demander l'établissement d'une connexion (*CONN.req*).

Une fois que la connexion avec le serveur a été établie, l'utilisateur peut introduire n'importe quelle commande du protocole SOLO. C'est bien là l'intérêt d'avoir un protocole en mode texte (ASCII).

Pour commencer, l'utilisateur peut s'authentifier vis-à-vis du serveur avec l'opération suivante (introduite pour Forum Lookup) :

```
12345?AUTH simple <interne,e3x,fr> interne
```

Si l'identité de l'utilisateur est connue et si le mot de passe qu'il a fourni est correct, le serveur le signalera par le message suivant :

```
12345:600 Ready.
```

L'utilisateur peut alors consulter le contenu de n'importe quelle entrée. On ne va considérer que deux requêtes de consultation dans cet exemple de dialogue :

```
12350?SOLO <worman, osi, e3x, fr> ? Phone, Email;

12350:202-No such name: <worman, osi, e3x, fr>
12350:301-Partial Match: <osi, e3x, fr> <OU=osi,O=e3x,C=fr>
12350:400 Suggestion: <CN=Ascan Woermann,OU=osi,O=e3x,C=fr>
```

```
12348?SOLO <woermann, osi, e3x, fr> ? Phone, Email;

12348:500 Matches: <CN=Ascan Woermann,OU=osi,O=e3x,C=fr>
Phone: +33 93.95.40.69
Email: woermann@sophia.telis-sc.fr
.
```

L'utilisateur a obtenu la réponse qu'il attendait : les moyens de contacter Ascan Woermann. Il ne lui reste alors plus qu'à se déconnecter du serveur :

```
12364?QUIT
```

7. Applications de mise en route

7.1 Introduction

Au cours du stage, on nous a demandé la réalisation de trois projets de difficulté croissante dans le cadre de Forum Lookup. Sans entrer à ce stade dans les détails, nous pouvons dire qu'il s'agissait de :

- réaliser un outil de test du bon fonctionnement du service offert par un ou plusieurs serveurs SOLO;
- réaliser un outil de test de la cohérence d'une base de données de l'annuaire;
- adapter le DSA à la programmation "*multi-thread*".

Remarquons que les deux premiers projets avaient pour objectif la réalisation de produits avec un cycle de vie complet (spécifications, réalisation, tests et gestion des versions) tandis que le troisième ne réclamait que la réalisation d'un prototype et de tests de performance.

Dans ce chapitre, nous allons nous concentrer sur les deux premiers projets. Le troisième, lui, sera complètement détaillé dans le chapitre 9.

7.2 SOLO-Ping

7.2.1 Présentation

L'utilitaire SOLO-Ping a pour but de vérifier le bon fonctionnement du service offert par un ou plusieurs serveurs SOLO (que ce soit un DUA-serveur, un SOLO-routeur ou un SOLO-serveur).

Par bon fonctionnement, on entend le fait de toujours recevoir un résultat à toute requête correcte que l'on a émise. A l'opposé, mauvais fonctionnement signifie ne pas recevoir de réponse de la part du serveur (endéans une certaine limite de temps) ou recevoir un message d'erreur de service (ce qui correspond à une incapacité à fournir le service d'annuaire).

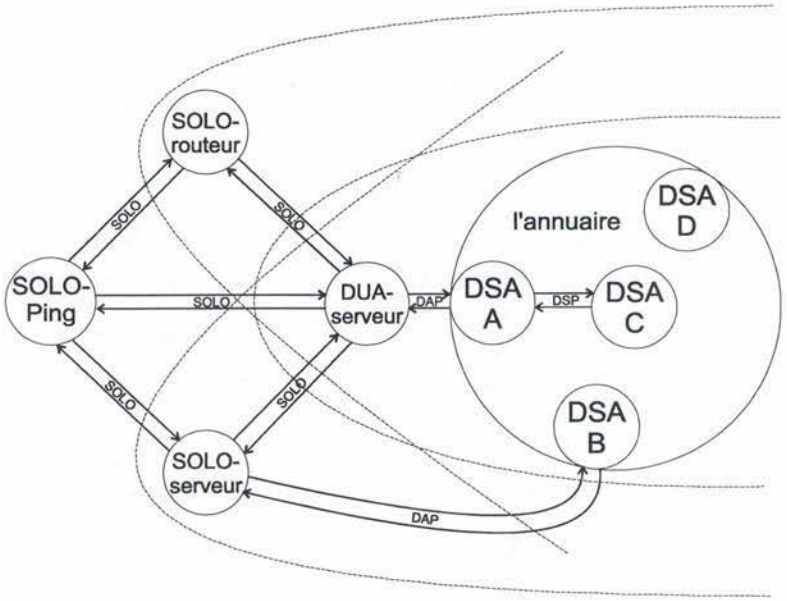


Figure 25 - SOLO-Ping et l'architecture logique de Forum Lookup

A la Figure 25, nous illustrons, pour chaque type de serveur SOLO auquel SOLO-Ping peut s'adresser, l'ensemble des serveurs de l'architecture Forum Lookup qui sont directement ou indirectement impliqués dans les tests effectués par SOLO-Ping. A cet effet, des courbes de tirets délimitent visuellement les serveurs concernés par les tests de SOLO-Ping.

Remarquons que tous les serveurs "encapsulés" dans la courbe du serveur SOLO testé ne sont pas dans l'obligation de fonctionner correctement. En effet, si l'on se reporte à l'architecture physique de Forum Lookup et au rôle de ses composants, l'on se rappellera que :

- dans la plate-forme Forum Lookup, il y a deux DSA qui gèrent exactement les mêmes entrées;
- si l'un des deux DSA de la plate-forme est hors service, le DUA-serveur bascule sur l'autre DSA de façon transparente.

Donc, SOLO-Ping ne teste pas le bon fonctionnement de tous les serveurs "encapsulés" dans la courbe mais bien la capacité qu'ont les serveurs SOLO qu'il interroge à fournir le service attendu.

7.2.2 Spécifications externes

Pour chaque serveur SOLO à tester, SOLO-Ping effectue les opérations suivantes :

1) ouverture d'une connexion avec le serveur;
2) envoi d'une requête d'authentification au serveur;
3) envoi d'une requête de consultation;
4) fermeture de la connexion.

Les PSAP des serveurs, les paramètres d'authentification et l'entrée à consulter sont définis dans un fichier de configuration.

SOLO-Ping peut être utilisé de deux façons différentes :

- mode "exécution unique" :
  Dans ce cas, le cycle "ouverture - requêtes - fermeture" n'est exécuté qu'une seule fois pour chaque serveur;
- mode "boucle infinie" :
  Dans ce cas, chaque serveur est interrogé successivement, puis un intervalle de temps (paramétrable par l'utilisateur) s'écoule avant de recommencer le cycle de tests.

7.2.3 Paramètres de configuration

SOLO-Ping lit le fichier de configuration contenant les paramètres `ServerPSAPList`, `UserName`, `UserPasswd`, `Entry`, `Attributes` et `Tmout`.

```
SOLO-PING
#--------
ServerPSAPList: localPureTCP=uranus+2220,localPureTCP=toscana+2228
UserName: interne,e3x,fr
UserPasswd: interne
Entry: heuse,e3x,fr
Attributes: CN,Email
Tmout: 10
```

- le paramètre `ServerPSAPList` permet de préciser les PSAP des serveurs à interroger;
- le paramètre `UserName` indique le nom de l'utilisateur qui sera envoyé au serveur dans la requête d'authentification;
- le paramètre `UserPasswd` contient le mot de passe associé au nom de l'utilisateur déclaré dans l'entrée précédente;
- le paramètre `Entry` contient le nom de l'entrée de l'annuaire qui sera consultée pendant le test;
- le paramètre `Attributes` contient les attributs que l'on désire récupérer lors de la requête de consultation effectuée sur l'entrée de test;
- Enfin, le paramètre `Tmout` indique la valeur du "*time-out*" que l'on souhaite voir appliquée à chaque opération que l'utilitaire va exécuter (ouverture de connexion, requête d'authentification et requête de consultation). Si le "*time-out*" est dépassé pendant l'une de ces opérations, la connexion avec ce serveur trop lent sera fermée et l'utilitaire passera à l'étape suivante.

Dans l'exemple ci-dessus, SOLO-Ping est configuré pour envoyer la requête de consultation des attributs `CN` et `Email` de l'entrée `<C=fr; O=e3x; OU=osi; CN="Bernard Heuse">` aux serveurs SOLO situés l'un au port TCP `2220` de la machine "`uranus`" et l'autre au port TCP `2228` de la machine "`toscana`".

7.2.4 Ligne de commande

Il reste maintenant à préciser ce qu'il faut mettre sur la ligne de commande :

```
slping [-b filePath] [-w waitTime]
```

Le premier paramètre, `filePath` (facultatif), représente le nom du répertoire contenant le fichier de configuration.

Le second paramètre, `waitTime` (aussi facultatif), indique le temps d'attente entre deux cycles "ouverture - requêtes - fermeture". Si ce paramètre est omis, un seul cycle sera effectué pour chaque serveur inscrit dans l'entrée `ServerPSAPList` du fichier de configuration. S'il est présent, le cycle "ouverture - requêtes - fermeture" sera répété indéfiniment et ce pour chaque serveur.

7.2.5 Evolution

Depuis la fin de notre stage, nous avons appris que SOLO-Ping a été légèrement modifié pour pouvoir consulter plusieurs entrées différentes dans un seul cycle "ouverture - requêtes - fermeture".

7.2.6 Exemple

Imaginons que l'on lance SOLO-Ping avec le fichier de configuration donné précédemment en exemple. Considérons, dès lors, qu'un SOLO-routeur se trouve sur la machine "`toscana`" à l'adresse `2228` et qu'un DUA-serveur se trouve à l'adresse `2220` sur la machine "`uranus`".

Cette situation est représentée par la Figure 26, où l'on a indiqué, de plus, que le SOLO-routeur était hors service.

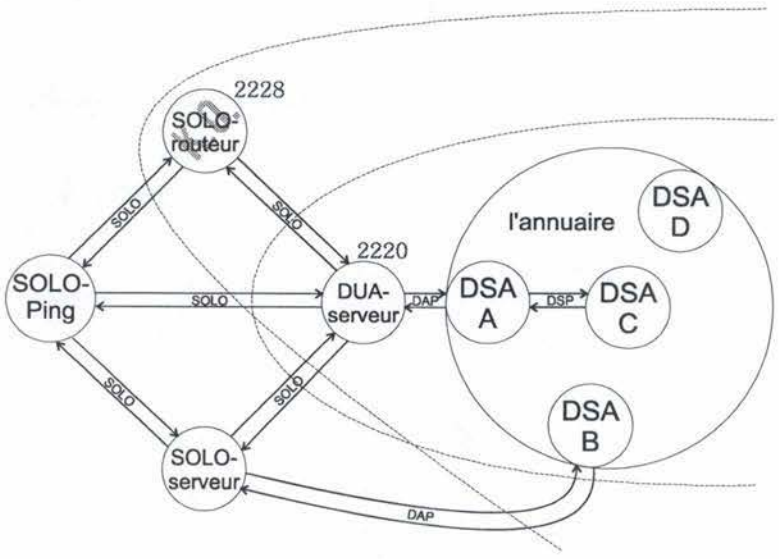


Figure 26 - Services offerts par des serveurs SOLO

Dans ce cas de figure, SOLO-Ping effectue les opérations suivantes :

1) ouverture d'une onnexion avec le DUA-serveur;

2) envoi d'une requête d'authentification au DUA-serveur;

3) envoi d'une requête de consultation au DUA-serveur;

4) fermeture de la connexion avec le DUA-serveur;

5) affichage d'un message d'erreur suite à l'échec rencontré lors de la tentative d'établissement d'une connexion avec le SOLO-routeur.

Si SOLO-Ping a été lancé en mode "exécution unique", il s'arrête à ce stade-ci. Sinon, il reprend (`waitTime` secondes plus tard) ses cycles de tests avec le DUA-serveur et de nouveau avec le SOLO-routeur.

Dans cet exemple, l'utilisateur de SOLO-Ping sait précisément isoler le serveur coupable du mauvais fonctionnement du service offert par le SOLO-routeur. Il sait arriver à la conclusion que c'est le SOLO-routeur lui-même qui est hors service vu qu'il effectue aussi un test du service offert par le DUA-serveur et que ce test est couronné de succès.

7.2.7 Aspects d'implémentation

La mise en œuvre de SOLO a été extrêmement simple : il nous a fallut environ deux semaines pour la réaliser entièrement (spécifications, conception, programmation et tests).

Nous n'avons rencontré aucune difficulté particulière lors du développement de cet utilitaire.

7.3 DB-Audit

7.3.1 Présentation

L'utilitaire DB-Audit effectue des tests de cohérence de la base de données de l'annuaire présentée sous forme textuelle. S'il découvre des erreurs, il les enregistre dans un fichier. Ainsi, l'administrateur peut corriger "manuellement" (avec un éditeur) la base de données erronée.

Une fois le contenu de la base de données textuelle vérifié (et éventuellement corrigé), la base peut être compilée en BER (avec l'utilitaire dbinit, dont nous ne parlerons pas) pour ensuite être consultée (et, éventuellement, modifiée) par un DSA.

Ce cycle d'édition, de correction et de compilation d'une base de données textuelle est illustré à la Figure 27.

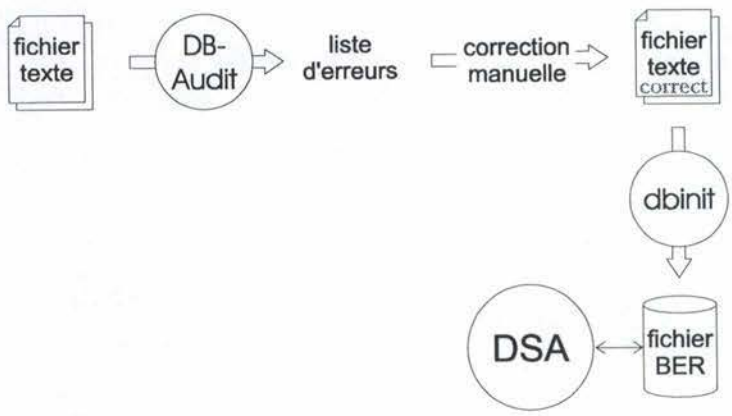


Figure 27 - Construction d'une base de données en BER

7.3.2 Spécifications externes

DB-Audit effectue les tests suivants sur la base de données :

- vérification syntaxique des entrées;
- vérification du schéma :
  Tous les attributs obligatoires doivent être présents dans toute entrée de la base et tous les autres attributs trouvés doivent être des attributs facultatifs;
- vérification de l'absence de doublons (entrées définies plus d'une fois);
- vérification de l'existence des entrées référencées dans les attributs dont les valeurs sont des DN (par exemple, dans les attributs `See-also`, `Alias`, `Manager` et `Secretary`);
- vérification de la cohérence des nomenclatures :
  Une **nomenclature** est définie comme étant la liste de toutes les valeurs d'un attribut répertoriées dans un sous-arbre donné. Concernant les nomenclatures, DB-Audit vérifie que :
  - ♦ toutes les valeurs d'attribut présentes dans une nomenclature d'un sous-arbre apparaissent dans les entrées de ce sous-arbre,
  - ♦ toute valeur présente dans une entrée d'un sous-arbre apparaît dans la nomenclature correspondante de ce sous-arbre.

Le concept de nomenclature est propre à Forum Lookup. Il a été introduit en particulier pour les IHM.

A l'heure actuelle, les nomenclatures `LocalityList` et `FunctionsList` sont des attributs facultatifs des entrées de classe "*Organization*". Elles permettent aux IHM d'afficher, après consultation de cette seule entrée, la liste de tous les lieux de travail et de toutes les fonctions (secrétaire, ingénieur, directeur, ...) des employés de l'entreprise. Le but de cette manoeuvre est d'autoriser une recherche de tous les employés qui travaillent dans une certaine localité ou qui ont une certaine responsabilité en empêchant l'utilisateur d'entrer des valeurs qui ne correspondent à aucun employé de l'entreprise.

Dans l'exemple proposé à la Figure 28, la nomenclature `FunctionsList`, stockée dans l'entrée `<C=fr; O=e3x>`, reprend la liste de toutes les valeurs de l'attribut `Title` dans les entrées subordonnées à `<C=fr; O=e3x>`. Elle permet ainsi aux IHM de proposer une recherche d'employés ayant une certaine fonction en donnant à l'utilisateur toutes leurs fonctions possibles et rien qu'elles.

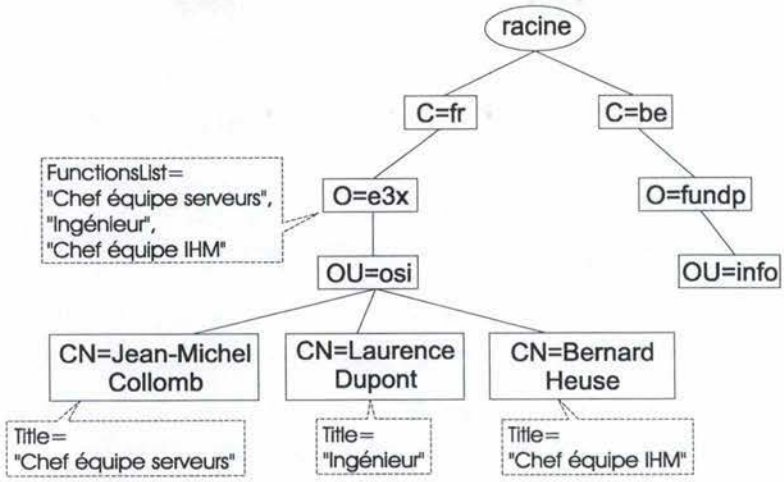


Figure 28 - Nomenclature FunctionsList

7.3.3 Paramètres de configuration

DB-Audit consulte au démarrage un fichier de configuration contenant les paramètres `RefAttribs` et `Lists`.

```
DB-AUDIT
#-------
RefAttribs: Alias, Manager, Secretary
Lists: LocalityList=L, FunctionsList=Title|CN(organizationalRole)
```

- le paramètre `RefAttribs` contient la liste des noms d'attribut dont les valeurs sont des références vers d'autres entrées de la base de données;
- le paramètre `Lists` contient la liste des nomenclatures à prendre en considération, ainsi que la façon dont elles doivent être construites.

Dans l'exemple ci-dessus, DB-Audit est configuré pour savoir que les attributs `Alias`, `Manager` et `Secretary` ont pour valeur des "pointeurs" vers des entrées de la base de données. DB-Audit devra donc consulter ces attributs-là (si l'utilisateur désire la vérification de l'existence des entrées référencées).

En outre, le paramètre `Lists` indique qu'il faut comparer le contenu de la nomenclature `LocalityList` de tout sous-arbre qui la possède avec l'ensemble des valeurs de l'attribut `L` des entrées de ce sous-arbre. De même, il faut comparer le contenu de la nomenclature `FunctionsList` avec l'ensemble des valeurs de l'attribut `Title` des entrées de ce sous-arbre et des valeurs de l'attribut `CN` des entrées de classe "*OrganizationalRole*" de ce même sous-arbre (restriction sur la portée de l'opération de comparaison).

7.3.4 Ligne de commande

Il faut introduire deux paramètres sur la ligne de commande :

```
dbaudit  [-b  filePath]  [-l 0|1|2|3]
```

Le premier paramètre, `filePath`, précise le chemin d'accès du fichier de configuration.

Le second paramètre indique quelles sont les vérifications que l'utilisateur souhaite effectuer :

- `0` correspond à la vérification syntaxique des entrées, à la vérification du schéma et à la vérification de l'absence de doublons;
- `1` correspond à l'ensemble des tests effectués par le niveau `0` avec, en plus, la vérification de l'existence des entrées référencées;
- `2` correspond à l'ensemble des tests effectués par le niveau `1` avec, en plus, la vérification du fait que les nomenclatures soient complètes;
- `3` correspond à l'ensemble des tests effectués par le niveau `2` avec, en plus, la vérification du fait que les nomenclatures soient minimales.

7.3.5 Exemple

La Figure 29 présente de façon visuelle une partie du contenu d'une base de données textuelle que l'on donne à vérifier à l'outil DB-Audit. Nous prévenons le lecteur que la figure est incomplète, puisque nous n'avons représenté aucun attribut obligatoire, entre autres. La figure attire néanmoins l'attention sur quatre types d'erreur et c'est dans cet optique qu'elle nous est utile.

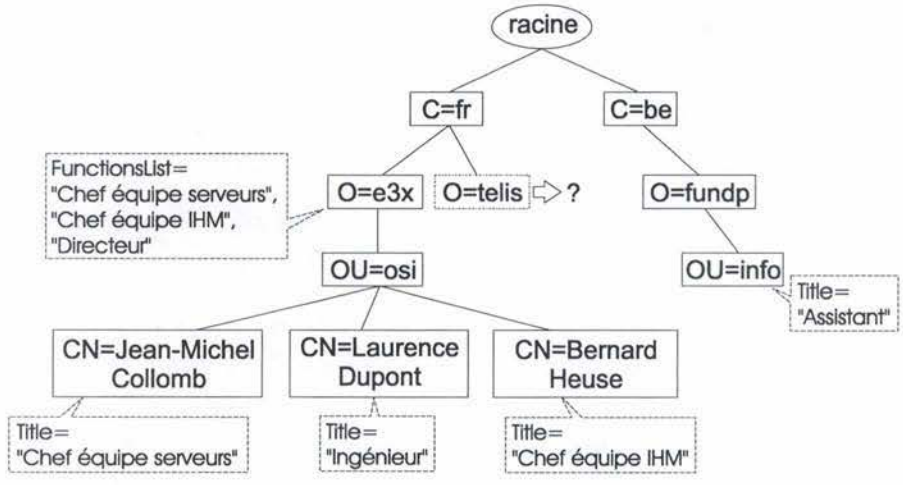


Figure 29 - Base de données dans un état incohérent

Enumérons les erreurs présentes dans cette base de données :

1) l'entrée `<C=be; O=fundp; OU=info>` contient l'attribut `Title` qui est, pourtant, interdit dans toute entrée de classe "*OrganizationalUnit*";
2) l'entrée alias `<C=fr; O=telis>` contient une référence vers une entrée inexistante (la référence étant, par exemple, le DN `<C=fr, O=telesystemes>`);
3) la nomenclature `FunctionsList` de l'entrée `<C=fr; O=e3x>` ne contient pas la valeur "`Ingénieur`" présente dans l'entrée subordonnée `<C=fr; O=e3x; OU=osi; CN="Laurence Dupont">`;
4) la nomenclature `FunctionsList` de l'entrée `<C=fr; O=e3x>` contient la valeur "`Directeur`" qui n'est présente dans aucune entrée subordonnée.

Pour chaque niveau de vérification permis par DB-Audit, nous allons indiquer les erreurs que notre outil découvre :

- le niveau `0` permet la découverte de la première erreur;
- le niveau `1` permet la découverte des deux premières erreurs;
- le niveau `2` permet la découverte des trois premières erreurs;
- le niveau `3` permet la découverte des quatre erreurs mentionnées ci-dessus.

7.3.6 Evolution probable

Il est tout à fait envisageable que, dans un futur proche, DB-Audit puisse lui-même corriger les erreurs qu'il découvre (en adoptant certains comportements qui seraient paramétrables par l'utilisateur).

Cette extension des capacités de DB-Audit est illustrée à la Figure 30.

Figure 30 - Nouvelle fonctionnalité de DB-Audit

7.3.7 Aspects d'implémentation

La réalisation d'une première version de DB-Audit nous a pris six semaines (spécifications, conception et programmation).

Lors de nos premiers tests, nous avons constaté que le temps pris par DB-Audit pour analyser une base de données de 75.000 entrées était d'environ 28 minutes. La taille du processus en mémoire était, en outre, trop élevée.

L'objectif qui nous était assigné étant d'analyser une base de 75.000 entrées en moins de 20 minutes, nous avons pris deux semaines supplémentaires pour optimiser DB-Audit.

La deuxième et dernière version de DB-Audit ne mettait plus que 18 minutes pour analyser une telle base de données et s'allouait beaucoup moins de mémoire qu'auparavant. L'objectif était donc pleinement satisfait.

8. Programmation "multi-thread"

8.1 Introduction

La programmation "*multi-thread*" constitue un paradigme de plus en plus répandu à l'heure actuelle. Les "*threads*" sont appropriés pour la plupart des programmeurs qui désirent exploiter du "*hardware*" parallèle (en vue d'améliorer les performances de leurs applications) ou exprimer la structure concurrente de leurs programmes avec plus de facilité.

Ce chapitre analyse les "*threads*" proposés par SunOS 5.0. Ils constituent un sous-ensemble du standard connu sous le nom de "*threads*" **UI** ("*Unix International*").

D'autres spécifications de "*threads*" existent, notamment la spécification 1003.1c (anciennement 1003.4a) de **POSIX** ("*Portable Operating System for unIX*"). Bien que nous ne l'aborderons pas, nous prétendons que cette dernière a de fortes chances de s'imposer comme standard dans un futur proche.

8.2 Définitions

Un "***thread***" (ou "fil", en français) est une séquence d'instructions exécutée dans un programme. Un "*thread*" possède [SUNS-92b] :

- un identifiant;
- un registre d'état, incluant un compteur de programme ("*P-Counter*") et un pointeur de pile;
- une pile (pour stocker les variables locales et les adresses de retour des fonctions appelées par le "*thread*");
- un masque de signaux;
- une priorité d'exécution (utilisée lors de la sélection d'un "*thread*" à exécuter);
- un **espace de stockage local de données** ("*Thread-Specific Data*" ou **TSD**).

A tout instant, il n'y a qu'un seul point d'exécution dans un "*thread*". Ce point d'exécution est donné par la valeur du "*P-Counter*".

Un processus "***multi-thread***" est un processus contenant plusieurs "*threads*" qui partagent un espace d'adressage commun et la plupart des autres ressources du processus.

8.3 Notions de base

Dans un processus "*multi-thread*", les "*threads*" doivent être créés et détruits de façon explicite. Une fois qu'ils sont créés, les "*threads*" s'exécutent indépendamment les uns des autres. En conséquence, l'exécution d'un tel processus est dite non déterministe car il n'y a en général aucun moyen de deviner comment les instructions exécutées par les différents "*threads*" vont s'entrelacer [SUNS-91].

Sur les machines à plus d'un processeur, les "*threads*" peuvent s'exécuter en parallèle sur les différents processeurs. Cela permet à une application unique de tirer profit de plusieurs processeurs, s'ils sont disponibles. Il peut donc y avoir simultanément plusieurs points d'exécution dans un processus "*multi-thread*".

Les "*threads*" font indépendamment des appels au système d'exploitation. Quand un "*thread*" attend la terminaison d'un service du système d'exploitation, il n'empêche pas les autres "*threads*" du processus de s'exécuter. Cela permet à une application unique de chevaucher efficacement les opérations d'**E/S** (**Entrée/Sortie**), comme le montre la Figure 31 [SUNS-93b].

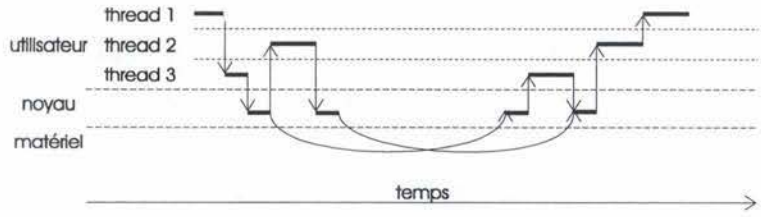


Figure 31 - Application "multi-thread"

Un processus ordinaire, lui, lorsqu'il envoie une requête au système d'exploitation, doit attendre la réponse à cette requête avant d'émettre la requête suivante. La Figure 32 [SUNS-93b] illustre cette séquence.

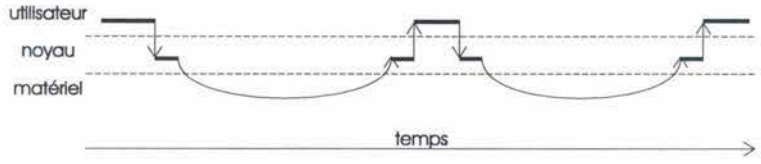


Figure 32 - Application ordinaire

Puisque les "*threads*" ont en commun un seul espace d'adressage (celui du processus), ils partagent les instructions du processus et la plupart de ses données [SUNS-91]. Par conséquent, un changement effectué par un "*thread*" dans une donnée partagée peut être vu par tous les autres "*threads*" du processus.

Les "*threads*" partagent aussi une grosse part de l'état du processus au niveau du système d'exploitation. Par exemple, si un "*thread*" ouvre un fichier, un autre peut le lire. Chaque "*thread*" voit ainsi les mêmes fichiers ouverts.

Puisque les "*threads*" partagent tant de l'état du processus, ils peuvent parfois s'affecter les uns les autres de façon surprenante (le système n'établissant aucune protection entre "*threads*"). Ainsi, certaines opérations affectent tous les "*threads*" d'un processus. Par exemple, si un "*thread*" appelle `exit()`, tous les "*threads*" sont détruits.

A part cela, chaque "*thread*" peut interagir avec d'autres processus de façon habituelle.

8.4 Architecture "multi-thread" de SunOS 5.0

Le modèle de programmation "*multi-thread*" de SunOS 5.0 comporte deux niveaux similaires mais essentiels [SUNS-91].

8.4.1 "Threads" utilisateur

Au niveau le plus important se trouvent les "*threads*" utilisateur (ou, plus simplement, les "*threads*") qui définissent l'interface principale du programmeur pour la programmation "*multi-thread*". Ces "*threads*" sont implémentés par une librairie en utilisant les "*lightweight processes*" (que nous abordons au point suivant). Ils sont inconnus du noyau.

Le programmeur peut voir le nombre de "*threads*" utilisés par son application comme le degré de parallélisme logique qu'elle réclame.

8.4.2 "Lightweight processes"

Au second niveau se trouvent les **processus légers** ("*LightWeight Processes*" ou **LWP**) sous-jacents aux "*threads*". Ils exécutent les "*threads*" du processus. Chaque LWP peut être vu comme un **CPU** ("*Central Processing Unit*") virtuel capable d'exécuter du code ou des appels système.

Le programmeur peut voir le nombre de LWP utilisés par son application comme le degré de concurrence réelle qu'elle réclame (parallélisme supporté par le noyau).

Cette seconde interface est définie pour les situations qui requièrent plus de contrôle sur la façon dont les programmes sont organisés sur le "*hardware*" parallèle et pour optimiser les coûts d'exécution concurrente et de synchronisation (nous reviendrons plus loin sur la synchronisation). Il est parfois important de savoir que cette interface existe même si peu de programmeurs l'utilisent directement.

En définissant deux niveaux d'interface dans l'architecture, une distinction claire est faite entre ce que voit le programmeur et ce que lui offre le noyau.

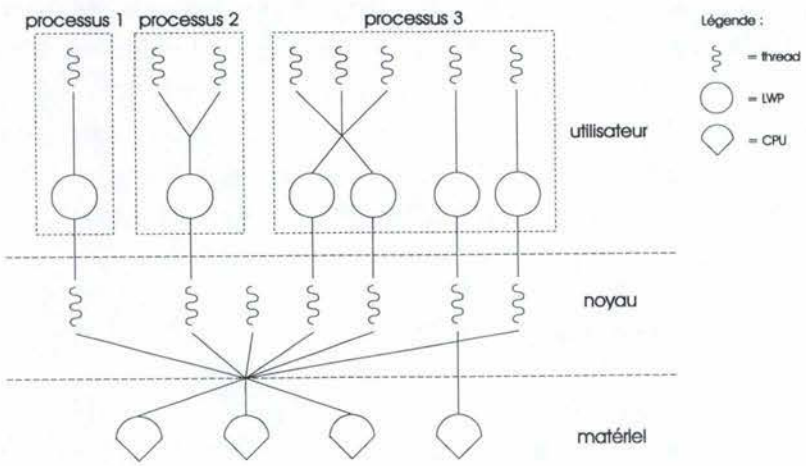


Figure 33 - Architecture "multi-thread" de SunOS 5.0

Dans la Figure 33 [SUNS-92a], le processus utilisateur 1 est le processus traditionnel UNIX avec un seul "*thread*" attaché à un seul LWP. Le processus 2 a deux "*threads*" multiplexés sur un seul LWP. Le processus 3 a plusieurs "*threads*" multiplexés sur un plus petit nombre de LWP ainsi que des "*threads*" attachés à des LWP.

8.4.3 "Threads" noyau

Le noyau de SunOS 5.0 [SUNS-92a] a été structuré autour des "*threads*" : il est lui-même un processus "*multi-thread*" très complexe.

Le noyau supporte l'exécution de LWP en associant un "*thread*" noyau à chaque LWP. Alors que tous les LWP sont supportés par un "*thread*" noyau, tous les "*threads*" noyau ne supportent pas de LWP.

8.4.4 Sélection des "threads" utilisateur

Le "*thread*" à exécuter par un LWP est choisi d'après son état (bloqué, en exécution ou en attente d'exécution) et sa priorité. Une fois le "*thread*" choisi, le LWP exécute ses instructions. Si le "*thread*" ne peut plus continuer ou si d'autres "*threads*" devraient être exécutés à sa place, le LWP sauvegarde l'état du "*thread*" en mémoire et sélectionne un autre "*thread*" à exécuter. Passer ainsi d'un "*thread*" à un autre se passe sans que le noyau le sache [SUNS-92b]. Similairement, des "*threads*" peuvent aussi être créés, détruits, bloqués, activés, ... sans impliquer le système d'exploitation. Séparer les "*threads*" des LWP autorise donc la librairie à passer rapidement d'un "*thread*" à l'autre sans entrer dans le noyau. De plus, cela permet à un processus utilisateur d'avoir des milliers de "*threads*" sans anéantir les ressources du noyau.

Quand un "*thread*" a besoin d'accéder à un service système en accomplissant un appel au noyau, il le fait en utilisant le LWP qui l'exécute [SUNS-91]. Le "*thread*" qui réclame le service système reste attaché au LWP qui l'exécute jusqu'au moment où l'appel système est terminé.

Parfois, un "*thread*" particulier doit voir sa visibilité étendue au système et non plus limitée au processus [SUNS-93b]. C'est le cas, par exemple, quand un "*thread*" doit permettre des réponses en temps réel. Il doit pouvoir être sélectionné parmi toutes les autres entités en exécution du système (portée globale de la sélection). La libraire s'en accommode en permettant à un "*thread*" d'être attaché de façon permanente à un LWP. Même quand un "*thread*" est attaché à un LWP, c'est toujours un "*thread*" et il peut donc interagir ou se synchroniser avec d'autres "*threads*" du processus, attachés ou non.

8.4.5 Taille de la réserve de LWP

La plupart des programmeurs qui utilisent les "*threads*" ne pensent pas aux LWP. Mais, quand il devient approprié d'optimiser le comportement du programme, les programmeurs ont la capacité de régler au mieux la relation entre les "*threads*" et les LWP [SUNS-92b].

Par défaut, la librairie "*threads*" ajuste automatiquement la taille de la réserve de LWP utilisés pour exécuter des "*threads*" non attachés. Il y a deux exigences principales dans le choix de la taille de cette réserve :

- elle ne doit pas permettre au programme d'entrer en interblocage (ou blocage permanent, voir plus loin) à cause du manque de LWP;
- les LWP doivent être utilisés efficacement.

Néanmoins, la décision à prendre concernant le nombre de LWP qui devraient être créés pour exécuter les "*threads*" peut être spécifiée par le programmeur.

8.4.6 Sélection des LWP

Chaque LWP est "dispatché" séparément par le noyau, peut exécuter des appels système indépendants et peut s'exécuter en parallèle sur une machine multiprocesseur [SUNS-91]. Tous les LWP du système sont sélectionnés par le noyau sur les CPU disponibles suivant leur priorité, leur état et certains autres paramètres (que nous choisissons d'ignorer).

Les LWP sont implémentés par le noyau. Ils sont relativement plus coûteux que les "*threads*" utilisateur (puisque chacun d'entre eux utilise des ressources du noyau dédicacées).

8.4.7 Sélection des "threads" noyau

Un "*thread*" noyau est l'entité fondamentale qui est prévue et "dispatchée" sur l'un des processeurs du système [SUNS-92a].

Un "*thread*" noyau est très léger, ayant seulement une petite structure de données et une pile. Passer d'un "*thread*" noyau à un autre est donc très bon marché.

8.5 Mécanismes de synchronisation des "threads"

8.5.1 Motivation

Lorsqu'il est possible que plusieurs "*threads*" accèdent simultanément (en écriture ou en lecture) à un emplacement mémoire et qu'au moins un de ces "*threads*" écrit dans cet emplacement mémoire, on parle de **course aux données** (ou "*data-race*") [SUNS-93a].

Pour empêcher les courses aux données qui altèrent leur consistance, un programmeur peut utiliser des mécanismes de synchronisation entre "*threads*" (pour leur permettre de synchroniser leurs accès aux données partagées) d'un ou de plusieurs processus.

Chaque mécanisme de synchronisation a une structure de données (appelée **variable de synchronisation**) qui est allouée dans la mémoire globale et diverses fonctions de synchronisation. Ces fonctions de synchronisation ont différentes sémantiques pour supporter différentes fréquences d'utilisation et différents styles d'interaction [SUNS-91].

Nous donnons ici une liste non exhaustive des principaux mécanismes de synchronisation.

8.5.2 Verrou "mutex"

Un **verrou d'exclusion mutuelle** (appelé aussi verrou "*mutex*" ou "*mutex lock*") [SUNS-93b] permet de s'assurer que, à tout moment, au plus un "*thread*" exécute une section de code (appelée **section critique**) qui consulte ou modifie certaines données partagées. Nous utilisons des "*mutexes*" pour limiter l'accès à une ressource partagée à un seul "*thread*" à la fois, en sérialisant l'exécution des "*threads*" concurrents (et, donc, en synchronisant leurs accès).

Le verrou "*mutex*" fonctionne comme suit. Avant de lire ou d'écrire dans une variable partagée à laquelle est associé un "*mutex*", un "*thread*" tente de verrouiller le "*mutex*" de la variable. S'il réussit à le verrouiller, le "*thread*" accède à la variable et déverrouille ensuite le "*mutex*". Si un autre "*thread*" essaie d'accéder à la variable pendant que le premier y accède, le second "*thread*" est bloqué quand il essaie de verrouiller le "*mutex*". Quand le premier "*thread*" en a fini avec la variable et déverrouille le "*mutex*", le second "*thread*" est débloqué et emporte le verrou pour le "*mutex*", pouvant alors accéder à la variable partagée [OPEN-92].

Les principales fonctions de la librairie "*threads*" SunOS 5.2 à propos des "*mutexes*" sont les suivantes :

- `mutex_lock()` acquiert un verrou ou bloque le "*thread*" appelant si le verrou est déjà tenu;
- `mutex_trylock()` acquiert un verrou s'il est libre et renvoie une erreur si le verrou est déjà tenu;
- `mutex_unlock()` déverrouille un verrou. Cette fonction doit être appelée par le "*thread*" qui a acquis le verrou dont il est question.

Par défaut, il n'y a aucun ordre d'acquisition du verrou entre plusieurs "*threads*" en attente.

8.5.3 Variable de condition

Une **variable de condition** ("*condition variable*") [SUNS-93b] est une variable partagée (requérant donc l'utilisation d'un "*mutex*") utilisée pour attendre jusqu'à ce qu'une condition particulière soit satisfaite.

Ainsi, en utilisant une variable de condition, le "*thread*" A peut attendre que le "*thread*" B accomplisse une certaine tâche. Pour cela, le "*thread*" A "bloque" sur la variable de condition jusqu'à ce que le "*thread*" B "signale" la variable de condition, indiquant que la tâche particulière a été accomplie.

Cette méthode de synchronisation est utilisée pour des communications explicites entre "*threads*" [OPEN-92]. Ces communications sont néanmoins anonymes. En effet, le "*thread*" B ne sait pas nécessairement que le "*thread*" A attend après la variable de condition et le "*thread*" A ne sait pas que c'est le "*thread*" B qui le réveille de son attente sur la variable de condition.

Les principales fonctions de la librairie "*threads*" SunOS 5.2 à propos des variables de condition sont les suivantes :

- `cond_wait()` bloque jusqu'à ce que la condition soit signalée;
- `cond_timedwait()` bloque jusqu'à ce que la condition soit signalée ou jusqu'à ce qu'un certain temps se soit écoulé;
- `cond_signal()` signale la condition.

8.5.4 Sémaphore

Un **sémaphore** [SUNS-93b] est, conceptuellement, un compteur entier (positif ou nul) initialisé au nombre de ressources libres.

Les "*threads*" décrémentent atomiquement le sémaphore quand une des ressources devient occupée et l'incrémentent quand une des ressources redevient libre. Quand le sémaphore devient nul, indiquant qu'il n'y a plus aucune ressource libre, les "*threads*" qui essaient de décrémenter le sémaphore sont bloqués jusqu'à ce qu'il redevienne strictement supérieur à 0.

Les principales fonctions de la librairie "*threads*" SunOS 5.2 à propos des sémaphores sont les suivantes :

- `sema_post()` incrémente le sémaphore (permettant peut-être à un "*thread*" de se débloquer);
- `sema_wait()` bloque le "*thread*" appelant jusqu'à ce que le sémaphore soit strictement positif et le décrémente ensuite;
- `sema_trywait()` décrémente le sémaphore s'il est strictement positif et renvoie une erreur sinon.

Par défaut, il n'y a aucun ordre d'acquisition du sémaphore entre plusieurs "*threads*" en attente.

8.5.5 Verrou "plusieurs lecteurs / un écrivain"

Un **verrou "plusieurs lecteurs / un écrivain"** ("*readers / writer lock*") [SUNS-93b] autorise plusieurs "*threads*" à accéder simultanément en lecture seulement à une donnée partagée. Il accorde aussi à un seul "*thread*" un accès en écriture à la donnée tout en excluant tout lecteur. Ce type de verrou est souvent utilisé pour protéger les données qui sont lues plus souvent qu'elles ne sont modifiées.

Les principales fonctions de la librairie "*threads*" SunOS 5.2 à propos des verrous "plusieurs lecteurs / un écrivain" sont les suivantes :

- `rw_rdlock()` acquiert un verrou en lecture ou bloque le "*thread*" appelant si un écrivain détient le verrou;
- `rw_tryrdlock()` acquiert un verrou en lecture si aucun écrivain ne détient le verrou et renvoie une erreur sinon;

- `rw_wrlock()` acquiert un verrou en écriture ou bloque le "*thread*" appelant si un lecteur ou un écrivain détient le verrou;
- `rw_trywrlock()` acquiert un verrou en écriture si aucun lecteur et aucun écrivain ne détient le verrou et renvoie une erreur sinon;
- `rw_unlock()` déverrouille un verrou en écriture ou en lecture suivant la nature de sa détention.

Par défaut, il n'y a aucun ordre d'acquisition du verrou entre plusieurs "*threads*" en attente. Néanmoins, les implémentations biaisent généralement l'ordre d'acquisition de telle sorte qu'aucun écrivain ne se retrouve en situation de **famine** (c'est-à-dire qu'il n'arrive jamais à "prendre la main" à cause d'un trop grand nombre de lecteurs). Ainsi, l'implémentation actuelle tend-elle à favoriser un écrivain par rapport à un lecteur.

8.5.6 Routine thr_join()

Il y a un autre mécanisme de synchronisation qui n'est pas anonyme : la routine `thr_join()`. Elle permet à un "*thread*" d'attendre qu'un autre "*thread*" spécifique termine son exécution. Quand le second "*thread*" s'est achevé, le premier se débloque et continue son exécution. Contrairement aux cas précédents, la routine `thr_join()` n'est associée à aucune donnée partagée.

8.5.7 Problème courant

Un **verrou mortel** (ou "*deadlock*"), encore appelé **interblocage**, est un blocage permanent d'un ensemble de "*threads*" qui sont en concurrence pour un ensemble de ressources. Ce problème est l'un des problèmes que l'on rencontre assez souvent.

L'erreur la plus courante à l'origine d'un verrou mortel consiste, pour un "*thread*", à essayer d'acquérir un verrou qu'il détient déjà. On parle alors de verrou mortel récursif. C'est le cas lorsque des fonctions que l'on a écrites doivent acquérir un verrou donné pendant toute la durée de leur appel et que ces fonctions s'appellent entre elles.

D'autres types de verrou mortel peuvent se produire entre deux ou plusieurs "*threads*". Par exemple, supposons que le "*thread*" A a acquis le verrou 1 tandis que le "*thread*" B a acquis le verrou 2. Si le "*thread*" A cherche alors à acquérir le verrou 2 et que le "*thread*" B essaie d'acquérir le verrou 1, les deux "*threads*" sont en interblocage.

8.6 Ecriture d'une librairie ré-entrante

Une routine est dite **ré-entrante** (ou "*MT-safe*") si elle se déroule correctement quand elle est exécutée simultanément par plusieurs "*threads*". L'idée de base d'un code ré-entrant est que son comportement est prédictible et non catastrophique lorsqu'il est exécuté par plusieurs "*threads*" à la fois [SUNS-95].

A l'opposé, une routine est non ré-entrante si elle ne se protège pas elle-même contre la concurrence dans un environnement "*multi-thread*".

Nous allons exposer ici les principes qui permettent d'écrire une routine ré-entrante. Il est très difficile de spécifier cela mais, en revanche, il est facile d'en écrire une fois que l'on a compris ce que signifie être ré-entrant. Pour ce faire, nous parlerons successivement d'interface et d'implémentation ré-entrantes.

Pour rappel, l'**interface** d'une fonction est sa déclaration. Elle comporte son nom, le type de donnée de sa valeur de retour ainsi que l'ordre et le type de ses paramètres. L'**implémentation**, elle, est le code de la fonction.

8.6.1 Interface ré-entrante

Une interface est dite ré-entrante si elle satisfait les critères suivants [SUNS-93a] :

- l'interface peut être appelée simultanément par plusieurs "*threads*" et fonctionnera conformément à la sémantique de la fonction;
- l'appelant n'est pas obligé de faire une synchronisation explicite avant d'appeler la fonction pour protéger les données directement manipulées par l'interface;
- l'implémentation de l'interface doit être faite sans courses aux données.

Il est intéressant de noter que, contrairement à ce que l'on pourrait penser, le comportement d'une interface quand elle est appelée par plusieurs "*threads*" ne doit pas nécessairement être le comportement d'une quelconque de ses exécutions sérielles pour être ré-entrante. En d'autres termes, une interface peut être ré-entrante alors que le comportement d'une de ses exécutions concurrentes n'est pas **sérialisable**.

Pour illustrer cette affirmation, considérons une interface qui ajoute une série d'éléments à une liste. L'interface peut être considérée ré-entrante si elle ajoute correctement un par un tous les éléments dans la liste. Dans ce cas, des exécutions concurrentes de cette interface peuvent entrelacer les éléments d'une façon qui n'est pas reproductible par une exécution sérielle des mêmes appels.

Remarquons toutefois que si le comportement de toutes les exécutions concurrentes de l'interface est sérialisable, alors l'interface est ré-entrante. La sérialisabilité des exécutions concurrentes d'une interface est donc une condition suffisante mais pas nécessaire de ré-entrance.

Le deuxième critère permet à certaines interfaces d'être ré-entrantes même si une partie de la mémoire qu'elles manipulent n'est pas directement sous leur contrôle (mais est manipulée via un pointeur). Ainsi, l'interface `memcpy()` peut être ré-entrante alors que, si deux "*threads*" l'appellent sur une même partie de la mémoire, il y a une course aux données.

Il revient en effet à l'appelant de verrouiller de façon appropriée la mémoire qui n'est pas sous le contrôle direct de l'interface car il doit supposer que toute la mémoire référencée peut être modifiée (à moins que la modification ne soit explicitement interdite dans la documentation de l'interface).

A l'opposé de tout ce qui vient d'être dit, si une interface satisfait l'une des conditions suivantes, alors elle est non ré-entrante :

- elle retourne un pointeur vers un "*buffer*" ou une structure alloué statiquement;
- elle se sert d'un ensemble de données globales (généralement appelé "contexte") comme arguments implicites.

8.6.2 Implémentation ré-entrante

Le problème le plus courant qui empêche du code existant d'être ré-entrant [SUNS-95] est l'utilisation de données globales qui conservent de l'information entre les appels à diverses routines. Ce problème a plusieurs solutions qui dépendent du comportement que ces routines devraient avoir dans un environnement "*multi-thread*".

Un exemple de fonction qui n'est pas ré-entrante est celui d'une fonction chargée de récupérer un enregistrement dans un fichier via l' "*offset*" (déclaré globalement) de l'élément courant dans le fichier. Dans une application "*multi-thread*", plusieurs "*threads*" pourraient vouloir lire les enregistrements du fichier. A ce moment-là, la fonction ne serait pas ré-entrante parce que l' "*offset*" de l'élément suivant ne serait pas correctement maintenu (étant modifié à chaque lecture). Pour qu'elle le devienne, il faudrait que l' "*offset*" de l'élément suivant soit local à chaque "*thread*" alors que le fichier, lui, pourrait rester global. Ce raisonnement n'est évidemment correct que si tous les "*threads*" veulent voir le même fichier en entier.

Comme nous venons de le montrer par l'exemple précédent, le problème de base de l'écriture d'un code ré-entrant est d'identifier :

- les données qui sont propres à chaque "*thread*",
- les données qui sont globales à tous les "*threads*".

Déterminer cela dépend de l'usage du code. Ceci exige d'analyser le code pour comprendre parfaitement ce qu'il fait.

Il faut absolument se rendre compte que mettre des verrous autour de toutes les variables globales ne va pas rendre ré-entrant un code qui ne l'est pas. Ce qu'il faut faire, c'est transformer certaines variables globales en variables locales selon la sémantique du code (pour que le comportement attendu du code soit celui observé). Les véritables variables globales, elles, requièrent des mécanismes de synchronisation.

D'après [SUNS-93b], tout code ré-entrant a les caractéristiques suivantes :

- le code ne change aucun élément de la mémoire globale
  Une exception à cette affirmation est tolérée dans le cas où la modification d'une donnée globale se fait sous protection de verrous et où la sémantique du code l'admet;
- le code ne se réfère à un élément de la mémoire globale que dans certaines circonstances
  Les circonstances particulières sont les suivantes : soit la référence à un élément de la mémoire globale se fait sous la protection de verrous, soit le programmeur sait que la donnée globale qui est partagée n'est jamais modifiée;

- le code n'accède jamais à un fichier ou à un périphérique
  La seule exception à la règle qui vient d'être énoncée est la demande du statut d'un fichier. Tout autre accès à un fichier change son état.

8.6.3 Adaptation de code

8.6.3.1 Aucune modification à apporter

Dans quelques cas, les interfaces sont complètement ré-entrantes.

Dans d'autres cas, elles ne sont pas ré-entrantes mais elles peuvent être appelées en toute sécurité par des programmes "*multi-thread*" si de tels appels ne se font qu'à partir du "*thread*" initial (aucune exécution concurrente).

Dans tous ces cas (peu nombreux), le programmeur n'a aucun changement à apporter au code existant.

8.6.3.2 Adaptations à faire

Dans la plupart des cas où le programmeur doit adapter du code non ré-entrant, l'on se rend compte que tout verrouillage pour des données partagées peut être caché dans l'implémentation de la routine [SUNS-96].

Mais, parfois, c'est l'interface qui n'est pas ré-entrante. Par exemple, si l'interface retourne un pointeur vers une zone de donnée allouée statiquement, elle n'est pas ré-entrante car le "*buffer*" pourrait être écrasé par un second appel à l'interface.

Une solution [SUNS-96] est de mettre le "*buffer*" dans une zone de donnée spécifique au "*thread*" (TSD). Cela permet aux "*threads*" d'appeler l'interface de façon indépendante. Cette approche a, néanmoins, le désavantage qu'une zone de donnée doit être allouée pour chaque "*thread*" qui utilise cette interface.

Une approche alternative est de définir des nouvelles interfaces ré-entrantes à ces fonctions. Pour les fonctions qui retournent des pointeurs vers des zones de donnée allouées statiquement, l'interface peut être changée de telle façon que l'appelant passe en argument un pointeur vers de la mémoire où les résultats peuvent être stockés. Cette approche permet à la fonction appelante de gérer la mémoire requise de façon appropriée.

Certaines autres interfaces peuvent être rendues ré-entrantes mais les suppléments impliqués dans le fait de cacher le verrouillage nécessaire sous l'interface est trop grand. Dans ce cas, on peut définir deux nouvelles interfaces de verrouillage et de déverrouillage à appeler avant et après une ou plusieurs utilisations de l'interface standard. Ceci permet à l'application de contrôler la granularité du verrouillage qui convient à ses besoins (plutôt que de verrouiller et de déverrouiller implicitement à chaque utilisation de l'interface standard).

Dans tous les cas, les routines non ré-entrantes sont utilisables pour autant que les "*threads*" synchronisent leurs accès à ces routines, par exemple en acquérant un "*mutex*" global et en le tenant pendant tout l'appel de la fonction [SUNS-92a]. Ce "*mutex*" assure qu'une seule routine de ce type peut être active à tout moment. Une solution de ce type a néanmoins le désavantage d'affecter les performances.

8.7 Séparation et découpage des tâches

Les types d'utilisation des "*threads*" peuvent se répartir en deux catégories : la séparation des tâches et leur découpage [ROSE-93b].

8.7.1 Séparation des tâches

Dans le premier cas, on indique simplement quelles routines d'application vont s'exécuter dans des "*threads*" séparés. Une telle séparation des tâches s'applique par exemple à des routines qui gèrent des périphériques lents, répondent à des saisies utilisateurs ou effectuent des **appels de procédure distante** ("*Remote Procedure Call*" ou **RPC**).

Un exemple d'application susceptible d'accroître sa vitesse d'exécution en utilisant la séparation des tâches est constitué par un programme de jeu d'échecs qui met à profit un "*thread*" pour permettre au programme de "réfléchir" pendant le tour de son adversaire, calculant les options possibles pour son prochain coup en attendant que son adversaire joue.

8.7.2 Découpage des tâches

Dans le second cas, on peut parfois accélérer l'exécution de l'application en décomposant une tâche lente en modules plus petits et en les déroulant dans des "*threads*" séparés. Parmi les exemples de découpage des tâches à l'aide des "*threads*", nous pouvons citer l'exécution concurrente de sous-tâches ou l'utilisation de processeurs spécifiques.

Les opportunités de découpage de tâches sont habituellement plus difficiles à détecter que celles permettant une séparation des tâches.

8.8 Processus versus "threads"

Pour exploiter plusieurs processeurs simultanément, on peut aussi faire tourner des processus séparés sur les différents processeurs plutôt que d'utiliser les "*threads*". Comparons brièvement ces deux approches différentes [ROSE-93b].

Le démarrage de plusieurs "*threads*" est relativement semblable à la création de plusieurs processus fils à partir d'un processus père (`fork()`).

A la différence des processus, cependant, les "*threads*" utilisent le même espace d'adressage, ce qui leur permet de partager toutes les données statiques ou externes, ainsi que les fichiers ouverts et toutes les autres ressources globales qui appartiennent au processus. Cette caractéristique facilite l'implémentation d'un grand nombre d'algorithmes de programmation parallèle.

Les "*threads*" requièrent également moins de mémoire que les processus, et dans certains cas, consomment moins de temps au démarrage et lors des changements de contexte.

Toutefois, il convient de modérer cette comparaison en faveur des "*threads*" en signalant que si un incident se produit dans un "*thread*" (suite à une erreur de programmation), il met en danger toute l'application "*multi-thread*". A l'opposé, dans le cas d'un incident dans un des processus fils d'une application, seul un processus devient hors service.

8.9 Avantages / Inconvénients

Les bénéfices de la programmation "*multi-thread*" sont évidents :

- parallélisme accru sur les machines multiprocesseurs (exploitation de la puissance du "*hardware*");
- amélioration (dans la plupart des cas) de la performance des programmes sur les machines monoprocesseurs;
- capacité de mieux exprimer l'asynchronisme propre à beaucoup de problèmes (code simplifié pour des applications complexes).

Mais la programmation "*multi-thread*" comporte aussi plusieurs inconvénients de taille :

- exigence de beaucoup plus d'attention et de discipline que la programmation ordinaire;
- besoin de restructurer les applications (dans le cas d'une maintenance adaptative);
- surcroît de travail pour le programmeur;
- nouveaux types d'erreur (courses aux données, interblocages, famine, ...);
- débogage et tests de programmes concurrents très difficiles (à cause du caractère non-déterministe de l'ordre d'exécution des différents "*threads*").

En résumé, l'utilisation des "*threads*" peut apporter des avantages réels à certains types d'application mais il est recommandé de mettre en balance ces avantages avec la complexité supplémentaire qui en découle dans les programmes [ROSE-93b].

9. Evolution de l'architecture du DSA

9.1 Introduction

Le DSA de Forum Lookup v1.1 est mono-processus et synchrone (c'est-à-dire qu'il ne gère qu'une requête à la fois). Ceci implique que :

- dans le cas d'une machine multiprocesseur, un seul processeur est utilisé (pas de parallélisme au niveau du processus);
- un client ne reçoit pas de réponse à sa requête tant que toutes les requêtes en cours ne sont pas traitées (file d'attente des requêtes);
- on constate des problèmes de priorité du traitement des requêtes rapides (*Read*) par rapport aux requêtes complexes (*Search*);
- on est confronté à des problèmes de charge en terme d'utilisateurs simultanés.

Dans la Figure 34 qui illustre le DSA de Forum Lookup v1.1, on note le fait qu'il soit synchrone en lui accolant le symbole d'une file d'attente. Le DUA-serveur étant, lui, asynchrone en est dépourvu.

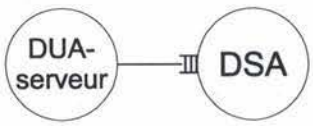


Figure 34 - DSA de Forum Lookup v1.1

Le dernier travail qui nous a été assigné lors de notre stage avait pour but de mettre en œuvre (un prototype à l'appui) une architecture "*multi-thread*" permettant de tirer profit des ordinateurs multiprocesseurs et de traiter simultanément plusieurs requêtes (asynchronisme).

Nous pouvons justifier ces deux objectifs de la façon suivante :

- les DSA de la plate-forme Forum Lookup étant hébergés sur des stations SUN Sparc 1000 (machines quadriprocesseurs), nous attachons de l'importance à l'exploitation parallèle des CPU.
- l'asynchronisme est, pour nous, un objectif tout aussi appréciable car nous estimons qu'il est plus important de démarrer la résolution des requêtes rapidement au prix d'une exécution un peu plus lente que de créer un retard au moment du lancement, en forçant les clients à patienter dans la file d'attente.

Avant de rentrer dans les détails de cette implémentation "*multi-thread*", nous allons d'abord examiner les différentes adaptations qui ont été faites au DSA pendant la période de notre stage.

9.2 Evolutions du DSA

9.2.1 DSA initial

Le DSA de Forum Lookup v1.1 est un seul processus UNIX apte à répondre aux requêtes des clients. D'un point de vue fonctionnel, il se divise néanmoins en deux composants distincts : un moteur protocolaire et un moteur X.500.

Le **moteur protocolaire**, appelé **DSA_DSA**, effectue les tâches suivantes :

- chaînage des requêtes (lorsque leur résolution ne peut pas être faite localement);
- concaténation des différents résultats lorsqu'il faut interroger plusieurs DSA (dans le cas d'un *Search* impliquant plusieurs sous-arbres gérés par différents DSA, par exemple).

Le **moteur X.500**, appelé **DSK_DSA** (DSK pour "*Directory Service Kernel*"), a pour but de résoudre les requêtes du service abstrait X.500. C'est lui qui traite les opérations *Read*, *Compare*, *List*, *Search*, *AddEntry*, *RemoveEntry*, *ModifyEntry* et *ModifyRDN*.

La division interne du DSA en moteur protocolaire et moteur X.500 est montrée à la Figure 35.

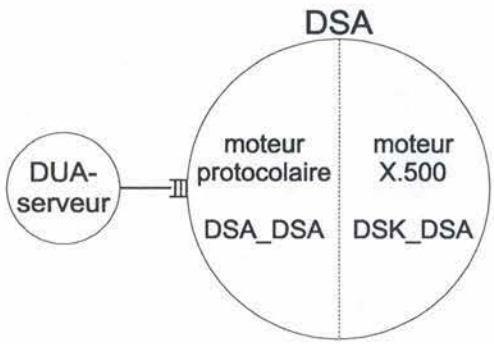


Figure 35 - Architecture mono-processus

La communication entre le moteur protocolaire et le moteur X.500 est implicite : il s'agit d'appels de routines avec passage de paramètres (structures C).

9.2.2 Première évolution

La première évolution qui a eu lieu a consisté à "couper" le DSA en deux processus UNIX distincts. La coupure s'est faite selon la division fonctionnelle qui a été mise en évidence dans le point précédent.

La communication entre le moteur protocolaire et le moteur X.500 est devenue explicite : il s'agit d'une communication entre processus selon un protocole propriétaire imaginé pour la cause. Cette communication est établie au niveau transport TCP ("*socket*" UNIX).

Le moteur X.500 dispose d'un port d'écoute qui lui permet de recevoir la connexion d'un client. Lorsqu'une demande de connexion du moteur protocolaire est reçue sur le port

d'écoute du moteur X.500, un port de service est créé et le moteur X.500 se met en attente de requêtes sur le port de service.

Pour s'échanger des informations, le moteur protocolaire et le moteur X.500 ne peuvent plus directement s'échanger des structures C. Pour passer sur la connexion TCP, les structures doivent être représentées sous la forme d'un flux de bits. Il faut donc les sérialiser avant de les envoyer et les "désérialiser" après les avoir reçues. MAVROS intervient notamment à ce stade en offrant au programmeur des fonctions d'encodage et de décodage de structures en BER (`XXX_cod()` et `XXX_dec()`).

La gestion de cette communication entre le moteur protocolaire et le moteur X.500 doit se faire dans les deux processus. En effet, le moteur protocolaire encode les requêtes qu'il envoie au moteur X.500 et décode les réponses qu'il reçoit de ce dernier. Le moteur X.500, lui, décode les requêtes qu'il reçoit du moteur protocolaire et encode les réponses qu'il lui envoie.

On pourrait se demander quelle est l'utilité d'un tel découpage du DSA puisque le service offert par ce DSA bi-processus n'est pas meilleur que le service offert par le DSA mono-processus. En fait, il y a quand même deux avantages à cette nouvelle architecture.

D'une part, le découpage du DSA en deux processus distincts est nécessaire pour passer à la deuxième évolution que nous allons détailler dans le prochain point.

D'autre part, le DUA-serveur peut retirer un avantage de cette découpe pour peu que son rôle ne se limite pas qu'à une simple traduction des requêtes SOLO en requêtes DAP. En effet, si on lui donne la liste des entreprises gérées par le DSA de Forum Lookup, il sera à même de décider si la requête pourra être résolue localement ou devra être chaînée vers d'autres DSA. Dans le premier cas (98 % des requêtes de consultation), il pourra s'adresser directement au moteur X.500 (évitant ainsi un intermédiaire inutile) pour autant qu'il sache communiquer avec lui selon le protocole propriétaire. Dans le second cas, il s'adressera normalement au moteur protocolaire.

Remarquons toutefois que, bien qu'il soit asynchrone, le DUA-serveur ne sait pas offrir à son client un service asynchrone à cause du DSK_DSA (qui, lui, est synchrone).

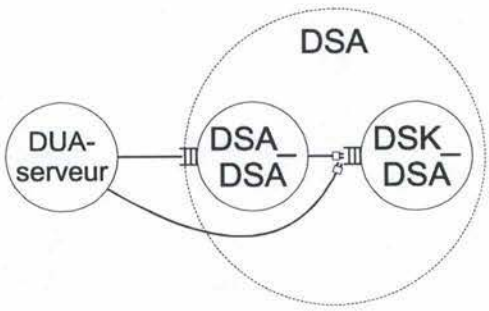


Figure 36 - Architecture bi-processus

La Figure 36 illustre ce nouveau DSA bi-processus. Le lien terminé par une prise de courant dessiné entre deux processus montre que la communication entre eux repose sur les "*sockets*" UNIX. La prise de courant sera toujours dessinée du côté du processus qui joue le rôle du serveur, c'est-à-dire celui auprès duquel il faut se connecter.

9.2.3 Deuxième évolution

L'évolution précédente n'a rien apporté au niveau du service offert. La deuxième évolution, elle, permet d'offrir un service globalement asynchrone par l'ajout d'un processus appelé le multiplexeur.

Le **multiplexeur** est un processus asynchrone qui fait le relais entre plusieurs moteurs X.500 et leurs clients. Il répartit les requêtes provenant du DSA_DSA et du DUA-serveur vers les différents moteurs X.500 en analysant leur charge.

Utilisé de manière transparente par le DSA_DSA et par le DUA-serveur, le multiplexeur permet de :

- répartir les charges sur les différents processeurs de la machine;
- utiliser en parallèle plusieurs moteurs X.500;
- fournir une meilleure disponibilité des moteur X.500 pour un plus grand nombre d'utilisateurs.

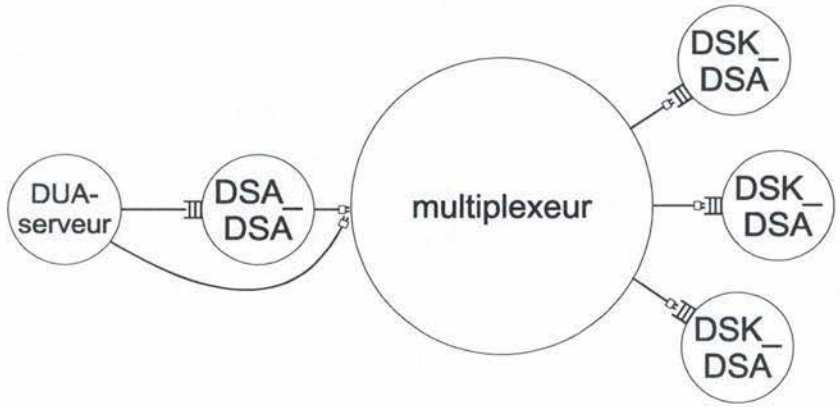


Figure 37 - Architecture avec multiplexeur

La Figure 37 illustre l'architecture résultant de la deuxième évolution.

Une variante de cette architecture consiste à considérer que certains des moteurs X.500 sont dédiés à certaines opérations. Par exemple, le premier moteur X.500 traite toutes les opérations du type *Read* et *Compare*, le deuxième toutes les opérations *List* et *Search* tandis que le troisième traite toutes les opérations de modification. Une telle architecture porte le nom d'architecture à **serveurs dédiés**.

Dans une architecture à serveurs dédiés, le multiplexeur a comme premier critère de sélection d'un moteur X.500 le type des opérations qu'il effectue et, comme second critère, sa charge.

L'avantage d'une telle architecture est de pouvoir éviter que tous les moteurs X.500 soient en train de traiter des requêtes complexes (*Search*) empêchant ainsi le traitement de requêtes rapides (*Read*).

C'est cette architecture qui est implémentée dans Forum Lookup v2.0.

9.2.4 Evolution vers le "multi-thread"

L'évolution qu'il nous a été demandé de réaliser lors du stage est une solution alternative à la deuxième évolution.

Une autre solution pour rendre un service globalement asynchrone consiste en effet à rendre le moteur X.500 asynchrone (en lui permettant de servir plus d'une requête client en même temps). Cela est possible de plusieurs façons, notamment en utilisant les "*threads*".

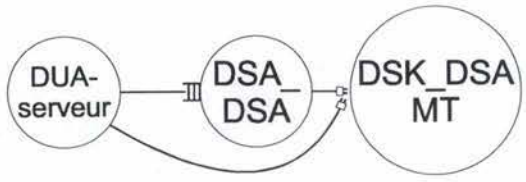


Figure 38 - Architecture "multi-thread"

La Figure 38 montre l'architecture du DSA lorsque le moteur X.500 est "*multi-thread*".

9.3 Avantages

Nous allons rapidement passer en revue les divers bénéfices de la solution "*multi-thread*" pour l'évolution du DSA de Forum Lookup.

Par rapport à la solution du multiplexeur, la solution "*multi-thread*" permet d'économiser toutes les communications entre le multiplexeur et les moteurs X.500. Cette économie, bien qu'elle ne soit pas considérable, n'est pas pour autant négligeable.

Par rapport à une autre solution, la solution "*multi-thread*" permet de bien structurer cette application asynchrone. Le code est, en effet, assez simple car il suffit d'écrire le traitement de chaque requête comme étant un "*thread*" indépendant et laisser la librairie gérer l'entrelacement des différentes opérations (interactions avec les clients, opérations relatives aux fichiers ou au réseau).

9.4 Contraintes

Adapter le DSA de Forum Lookup à la programmation "*multi-thread*" pose néanmoins quelques problèmes que nous allons présenter ici.

Les "*threads*" ayant des implications profondes sur la façon de concevoir une application, il est difficile de les intégrer a posteriori dans une application existante.

Comme nous l'avons déjà expliqué au chapitre 8, il faut, en particulier, rendre ré-entrantes toutes les routines du serveur "*multi-thread*". Elles doivent être adaptées afin de pouvoir s'exécuter concurremment en partageant les mêmes données globales.

Pour cela, il faut étudier (voire modifier) la structure des variables globales et la manière dont on y accède. En fonction de cela, il faut alors concevoir des verrous appropriés. Dans notre cas, cette étape a été la plus fastidieuse (car seul un verrouillage à granularité fine était acceptable, étant donné l'importance que nous attachions aux performances du serveur).

9.5 Analyse du problème

Comme nous l'avons déjà précisé précédemment, les types d'utilisation des "*threads*" peuvent se répartir en deux catégories : la séparation des tâches et leur découpage.

Dans le premier cas, on s'arrange pour traiter une requête client dans un "*thread*" indépendant. Par conséquent, le traitement de toute requête est séquentiel et le temps d'exécution d'une requête isolée n'est pas réduit.

Dans le deuxième cas, on accélère le traitement d'une requête client en le décomposant en sous-tâches plus élémentaires et en les déroulant dans des "*threads*" séparés.

Avoir pour objectif le découpage du traitement des requêtes aurait été déraisonnable en fonction du temps qui nous était imparti et de la difficulté supplémentaire à le mettre en œuvre dans une application existante.

Notre objectif a donc été d'adapter le code du DSA pour qu'il traite chaque requête dans un "*thread*" séparé.

9.6 Conception de la solution

9.6.1 Initialisation

Le serveur commence sa vie comme un seul "*thread*". A ce moment, il initialise des variables sans qu'aucun verrou ne soit nécessaire. Des courses aux données sont en effet impossibles puisqu'il est le seul "*thread*" à s'exécuter.

Plus précisément, l'initialisation consiste, pour lui, à charger en mémoire le DIT (formé à partir des entrées d'une base de données stockée sur disque), à créer des tables d'index (pour accélérer l'accès à certaines valeurs d'attribut) et à créer un nombre suffisant de descripteurs de fichier pour pouvoir accéder en parallèle à la base de données (fichier BER).

Le DIT en mémoire est agrémenté de quelques informations techniques dont nous ne parlerons pas, à l'exception de l' "*offset*" de chaque entrée dans la **BD** (**Base de Données**).

9.6.2 Gestion des demandes de connexion

Une fois l'initialisation terminée, le serveur s'occupe de recevoir les demandes de connexion de différents clients sur le port d'écoute dont il dispose.

Lorsqu'une demande de connexion d'un client est reçue sur le port d'écoute, un port de service est créé. Le serveur se remet alors en attente de nouvelles demandes de connexion sur le port d'écoute et en attente de requêtes sur le ou les ports de service.

9.6.3 Gestion des requêtes

Lorsqu'une requête arrive sur un port de service, le serveur crée un "*thread*" et se remet en attente sur le port d'écoute et les ports de service. Le "*thread*", lui, se charge de résoudre la requête et d'y répondre (sur le port de service du client qui l'a émise) avant de se détruire.

9.6.4 Tâche des "threads"

Le "*thread*" commence par analyser le type de l'opération qu'on lui demande d'effectuer. Il essaie ensuite d'acquérir un verrou du type "plusieurs lecteurs / un écrivain" en lecture (s'il s'agit d'une opération de consultation) ou en écriture (s'il s'agit d'une opération de modification). Une fois le verrou acquis, le "*thread*" traite la requête du client et libère finalement le verrou qu'il détient.

Cette façon de procéder permet au "*thread*" de garantir à tout moment la consistance du DIT en mémoire et la consistance de la BD sur disque.

Nous allons maintenant être plus explicite quant aux actions effectuées par le "*thread*" lors du traitement de la requête d'un client :

- dans le cas d'une requête de modification, le "*thread*" modifie éventuellement le DIT en mémoire et, dans tous les cas, rajoute à la fin de la BD une ou plusieurs entrées spéciales contenant les modifications qui ont été effectuées. Bien que nous pourrions approfondir le cas des opérations de modification, nous nous en passerons car nous considérons (à juste titre) que le nombre d'opérations de consultation est de loin supérieur au nombre d'opérations de modification. Dès lors, elles ne nous intéressent que de façon marginale;
- dans le cas d'une opération de consultation, le "*thread*" examine le DIT pour obtenir les "*offsets*" dans la BD des entrées qui l'intéressent. Cela lui permet d'accéder directement au contenu de ces entrées en utilisant l'un des descripteurs de fichier qui n'est utilisé par aucun autre "*thread*".

9.6.5 Consultations en parallèle

Le moteur X.500 peut résoudre plusieurs opérations de consultation en même temps grâce au parallélisme permis par le verrou du type "plusieurs lecteurs / un écrivain" et grâce à la duplication du descripteur de fichier relatif à la BD. Par conséquent, d'un point de vue macroscopique, le moteur X.500 peut accéder en parallèle à plusieurs entrées de la BD.

9.7 Tests de performance

9.7.1 Environnement

Toutes les mesures de performance indiquées dans ce mémoire ont été obtenues sous SunOS 5.3 sur une station SUN Sparc 1000 (équipée de 256 Mo RAM). Les mesures ont été faites en utilisant l'horloge interne de la station dont la résolution est la microseconde (us).

Précisons, d'ores et déjà, que les résultats de tous les tests de performance décrits dans ce texte sont présents en annexe (annexe B).

9.7.2 Objectif

L'objectif de ces tests de performance était d'évaluer les temps de réponse d'un DSA "*multi-thread*" mono-processus, illustré à la figure 39. Nous insistons sur le fait que nous voulions considérer un DSA mono-processus car la coupure en deux processus ne se justifie que dans le cadre particulier d'utilisation du DSA dans Forum Lookup et car nous voulions des résultats qui ne soient pas restreints à ce cadre.

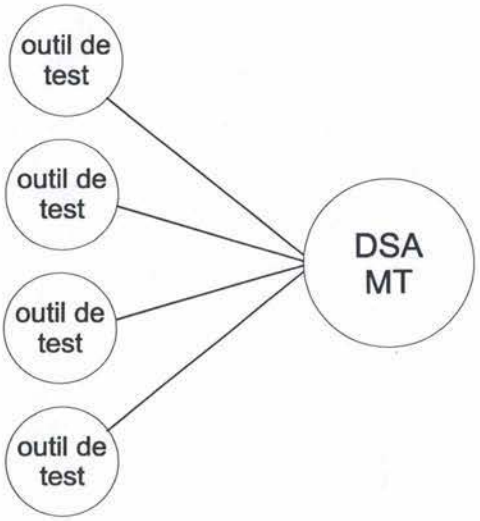


Figure 39 - DSA "multi-thread" mono-processus

Pour émuler ce DSA "*multi-thread*" mono-processus que nous n'avions pas à notre disposition, nous avons testé notre moteur X.500 "*multi-thread*" avec plusieurs moteurs protocolaires (il en faut plusieurs car ils sont synchrones). Ces derniers étaient eux-mêmes exploités par des outils de test qui simulaient un utilisateur en émettant séquentiellement 210 requêtes de consultation. Ces outils de test calculaient les temps séparant l'émission des requêtes et la réception des réponses. Ces relations sont illustrées à la Figure 40.

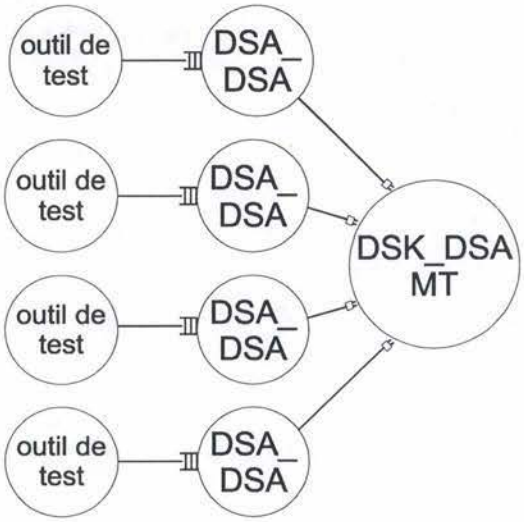


Figure 40 - DSA composé du moteur X.500 "multi-thread"

Pour que les temps de réponse calculés avec le moteur X.500 "*multi-thread*" représentent assez fidèlement les temps de réponse que l'on aurait obtenu avec le DSA "*multi-thread*" que nous imaginions avoir, il fallait cependant leur retrancher le temps impliqué par la communication entre le moteur protocolaire et le moteur X.500.

En effet, la communication entre moteurs implique le codage, le transfert et le décodage des requêtes ainsi que le codage, le transfert et, finalement, le décodage des réponses. Dans un but de simplification, appelons dorénavant "temps de communication entre moteurs" ce temps global de codages, de transferts et de décodages.

9.7.3 Outils

Les outils de test envoient en tout et pour tout 210 requêtes. Lors de nos tests, nous avons choisi de toujours émettre les mêmes requêtes, et ce, pour deux raisons. D'une part, cela permettait de lancer plusieurs fois les tests afin d'isoler les temps de réponse représentatifs (c'est-à-dire ceux qui sont constants au cours des différents tests). Et, d'autre part, cela permettait de comparer les différentes architectures face au même jeu de test.

Les requêtes générées par les outils de test sont classifiées comme suit :

- 10 requêtes différentes de type *Bind*;
- 10 requêtes différentes de type *Read*;
- 190 requêtes différentes de type *Search*, dont :
    - 70 requêtes ont leur portée limitée aux entrées directement subordonnées à l'entrée de base (30 recherches par égalité, 20 recherches par sous-chaîne et 20 recherches par approximation),
    - 120 requêtes ont leur portée étendue à tout le sous-arbre commençant à l'entrée de base (40 recherches par égalité, 40 recherches par sous-chaîne et 40 recherches par approximation).

Comme nous l'avions déjà annoncé, nous n'avons fait aucun test de performance incluant des requêtes de modification vu leur fréquence marginale d'apparition.

9.7.4 Temps de réponse pour un client isolé

Ce point détaille les tests effectués pour évaluer les temps de réponse du DSA "*multi-thread*" mono-processus lorsqu'il est soumis aux 210 requêtes d'un seul client.

9.7.4.1 Calcul du temps de communication entre moteurs

La première chose que nous avons faite a consisté à calculer le temps de communication entre moteurs. Nous l'avons tout simplement obtenu en comparant les temps de réponse aux requêtes d'un client dans le cas du DSA mono-processus synchrone (DSA initial) et dans le cas du DSA bi-processus synchrone (première évolution). La Figure 41 et la Figure 42 illustrent ces deux premiers tests que nous avons menés.

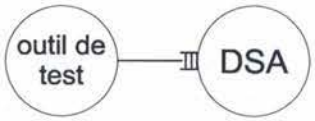


Figure 41 - Premier test

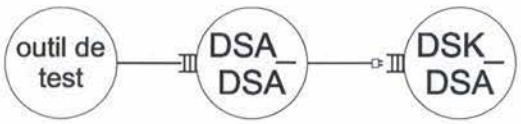


Figure 42 - Deuxième test

Les outils de test générant les mêmes requêtes, il nous était possible de calculer avec précision le temps de communication entre moteurs.

Ce temps, variable selon les requêtes, valait entre 20 et 116 ms pour des requêtes prenant entre 104 et 613 ms à être résolues par le DSA bi-processus. Il était donc loin d'être négligeable.

9.7.4.2 Temps de réponse du DSA "multi-thread" mono-processus

Suite au calcul du temps de communication entre moteurs pour chaque type de requête d'un client, nous avons calculé les temps de réponse du serveur composé du moteur X.500 "*multi-thread*" (voir Figure 43) lorsqu'il était soumis au même jeu de test.

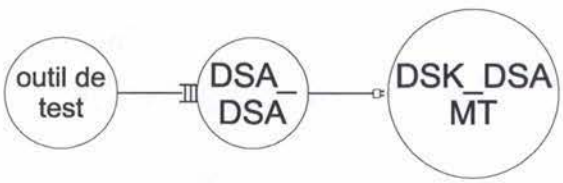


Figure 43 - Troisième test

A ces résultats, nous avons retranché le temps de communication entre moteurs pour évaluer les résultats du DSA "*multi-thread*" mono-processus que l'on imaginait ainsi avoir à notre disposition.

9.7.4.3 Comparaison avec le DSA mono-processus synchrone

Nous allons reprendre dans le tableau 1 les résultats les plus intéressants des différents tests que nous avons effectués (dans le cas d'un seul client).

Pour garder les mêmes notations que celles qui sont utilisées en annexe, nous avons repris (en première colonne) les noms que nous attribuons aux différents types d'opération. Cependant, nous n'expliciterons pas leur signification ici, les résultats bruts étant la seule chose qui nous intéresse. Le lecteur désirant une analyse plus fine en fonction des types de requête est prié de se reporter à l'annexe B.

Expliquons maintenant le contenu des quatre dernières colonnes :

- la première rassemble les temps de réponse du serveur composé du moteur X.500 "*multi-thread*";
- la deuxième rassemble les temps de communication entre moteurs, dans le cas d'un client;
- la troisième rassemble les temps de réponse du DSA "*multi-thread*" mono-processus, obtenus par soustraction des résultats de la première colonne avec ceux de la deuxième colonne;
- la quatrième, enfin, rassemble les temps de réponse du DSA mono-processus synchrone (DSA initial).

	1	2	3	4
Bi	0.12	0.01	0.11	0.10
Re	0.12	0.04	0.08	0.07
Se2=1(CN)	0.15	0.03	0.12	0.11
Se2&1(S)	0.21	0.05	0.16	0.14
Se2~1(S)	0.31	0.06	0.25	0.19
Se2=1(C/CN)	0.15	0.03	0.12	0.11
Se2&1(C/S)	0.24	0.05	0.19	0.17
Se2~1(C/S)	0.37	0.07	0.30	0.26
Se2=1(C/CN/T)	0.15	0.03	0.12	0.11
Se1=5(OU)	0.14	0.03	0.11	0.09
Se1~5(OU)	0.14	0.03	0.11	0.09
Se1&5(OU)	0.14	0.03	0.11	0.09
Se1=5(C/OU)	0.14	0.03	0.11	0.09
Se1~5(C/OU)	0.14	0.03	0.11	0.09
Se1&5(C/OU)	0.14	0.03	0.11	0.09
Se1=6(CN)	0.16	0.03	0.13	0.11
Se1~6(CN)	0.16	0.03	0.13	0.10
Se1&6(CN)	0.16	0.03	0.13	0.11
Se1=6(C/CN)	0.16	0.03	0.13	0.11
Se1~6(C/CN)	0.16	0.03	0.13	0.10
Se1&6(C/S)	0.17	0.03	0.14	0.11

Tableau 1 - Tests de performance pour un client

Pour chaque type de requête, les temps de réponse obtenus avec le DSA "*multi-thread*" mono-processus sont légèrement supérieurs à ceux obtenus avec le DSA mono-processus synchrone.

9.7.4.4 Interprétation

Le DSA "*multi-thread*" n'est donc pas très efficace dans le cas d'un client isolé. Mais cela est tout à fait normal étant donné qu'il n'y a aucun parallélisme.

En effet, rappelons que notre but n'était pas d'optimiser le temps de traitement d'une requête isolée mais bien le temps de traitement de requêtes en parallèle et, qu'à cet effet, nous n'avions pas choisi de découper le traitement d'une requête particulière dans des "*threads*" séparés.

9.7.5 Temps de réponse pour quatre clients simultanés

Ce point détaille les tests effectués pour évaluer les temps de réponse du DSA "*multi-thread*" mono-processus lorsqu'il est soumis aux requêtes simultanées de quatre clients.

9.7.5.1 Calcul du temps de communication entre moteurs

De manière similaire au cas précédent (où il n'y avait qu'un seul client), nous avons commencé par calculer les temps de réponse du DSA mono-processus synchrone et du DSA bi-processus pour obtenir une idée du temps de communication entre moteurs.

La Figure 44 et la Figure 45 illustrent ces deux tests.

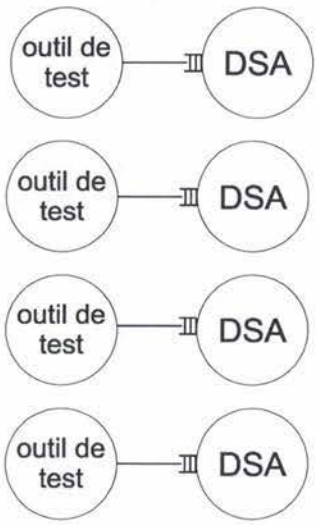


Figure 44 - Quatrième test

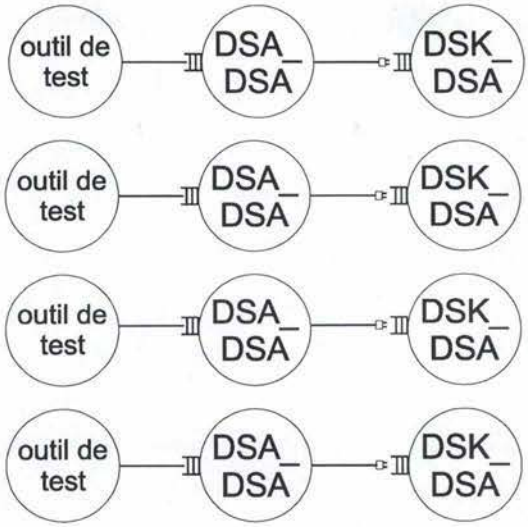


Figure 45 - Cinquième test

Ces deux tests nous ont permis de constater que le temps de communication entre moteurs était bien plus élevé lorsqu'il y avait quatre clients que lorsqu'il n'y avait qu'un client isolé.

Ce temps, variable selon les requêtes, valait entre 63 et 366 ms pour des requêtes prenant entre 170 et 790 ms à être résolues par le DSA bi-processus.

9.7.5.2 Temps de réponse du DSA "multi-thread" mono-processus

Après que nous ayons calculé le temps de communication entre moteurs, nous avons calculé les temps de réponse du serveur composé du moteur X.500 "*multi-thread*" avec quatre clients qui l'inondent de requêtes (voir figure 46).

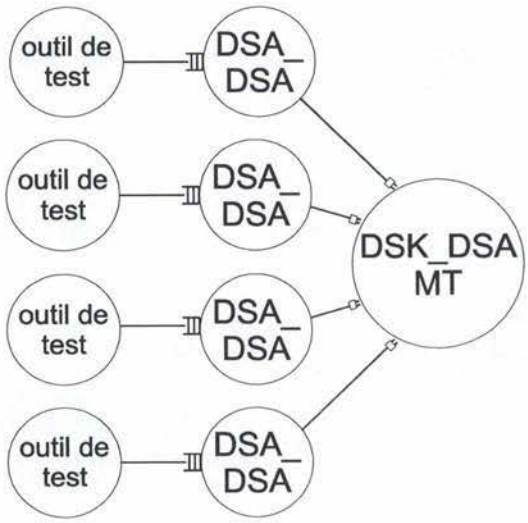


Figure 46 - Sixième test

A tous ces résultats, nous avons retranché le temps de communication entre moteurs que nous venions de calculer dans le point précédent.

9.7.5.3 Comparaison avec le DSA mono-processus synchrone

Dans le tableau 2, on retrouve les résultats les plus intéressants des différents tests que nous avons effectués (dans le cas de quatre clients simultanés).

Expliquons le contenu des six dernières colonnes :

- la première rassemble les temps de réponse du serveur composé du moteur X.500 "*multi-thread*";
- la deuxième rassemble les temps de communication entre moteurs, dans le cas de quatre clients;
- la troisième rassemble les temps de communication entre moteurs, calculés précédemment dans le cas d'un client isolé;
- la quatrième rassemble les temps de communication entre moteurs "raisonnables" (calculés en prenant la moyenne des deux précédentes colonnes);
- la cinquième rassemble les temps de réponse "raisonnables" du DSA "*multi-thread*" mono-processus, obtenus par soustraction des résultats de la première colonne avec ceux de la quatrième colonne;
- la sixième, enfin, rassemble les temps de réponse du DSA mono-processus synchrone (DSA initial).

Dans l'explication du contenu des colonnes, si nous utilisons le terme "raisonnable", c'est parce que nous nous sommes rendus compte que, dans le cas des quatre clients simultanés, les temps de communication entre moteurs que nous avons calculés étaient trop élevés. En effet, si l'on retranche ces temps de ceux de la première colonne, on en arrive avec des temps très petits, voire négatifs (dans certains cas).

Dès lors, dans le but de modérer ces résultats que nous jugeons excessifs, nous avons décidé de considérer, comme temps "raisonnables" de communication entre moteurs, la moyenne des temps de communication entre moteurs dans le cas d'un client isolé et dans le cas de quatre clients simultanés.

Nous pensons que cette hypothèse fondamentale nous permet d'obtenir des interprétations bien plus réalistes.

	1	2	3	4	5	6
Bi	0.17	0.00	0.01	0.00	0.17	0.18
Re	0.16	0.06	0.04	0.05	0.11	0.11
Se2=1(CN)	0.18	0.16	0.03	0.10	0.09	0.23
Se2&1(S)	0.33	0.24	0.05	0.14	0.19	0.31
Se2~1(S)	0.60	0.37	0.06	0.22	0.38	0.40
Se2=1(C/CN)	0.24	0.28	0.03	0.15	0.08	0.25
Se2&1(C/S)	0.45	0.28	0.05	0.16	0.28	0.39

Se2~1(C/S)	0.71	0.27	0.07	0.17	0.55	0.52
Se2=1(C/CN/T)	0.24	0.18	0.03	0.11	0.14	0.27
Se1=5(OU)	0.19	0.15	0.03	0.09	0.10	0.22
Se1~5(OU)	0.19	0.17	0.03	0.10	0.09	0.21
Se1&5(OU)	0.21	0.15	0.03	0.09	0.12	0.21
Se1=5(C/OU)	0.21	0.12	0.03	0.08	0.13	0.21
Se1~5(C/OU)	0.23	0.14	0.03	0.09	0.14	0.21
Se1&5(C/OU)	0.24	0.15	0.03	0.09	0.15	0.20
Se1=6(CN)	0.28	0.14	0.03	0.09	0.19	0.22
Se1~6(CN)	0.30	0.13	0.03	0.08	0.22	0.22
Se1&6(CN)	0.29	0.12	0.03	0.07	0.22	0.23
Se1=6(C/CN)	0.28	0.11	0.03	0.07	0.21	0.23
Se1~6(C/CN)	0.27	0.09	0.03	0.06	0.21	0.22
Se1&6(C/S)	0.25	0.08	0.03	0.05	0.20	0.23

Tableau 2 - Tests de performance pour quatre clients simultanés

Pour la plupart des types de requête, les temps de réponse obtenus avec le DSA "*multi-thread*" mono-processus sont inférieurs (parfois, même largement) à ceux obtenus avec le DSA mono-processus synchrone.

9.7.5.4 Interprétation

Le DSA "*multi-thread*" se montre assez performant dans le cas de quatre clients simultanés (grâce à la mise en œuvre du parallélisme).

Il est évidemment normal qu'il soit meilleur que le DSA mono-processus synchrone, qui ne sait répondre qu'à un client à la fois.

Cette tendance de supériorité du DSA "*multi-thread*" mono-processus par rapport au DSA mono-processus synchrone avait l'air de se confirmer de plus belle dans le cas de huit clients simultanés. Néanmoins, n'ayant pas eu l'occasion de tester en profondeur ce cas, nous n'en présenterons aucun résultat. Nous ne faisons que divulguer au lecteur notre intime conviction.

9.8 Aspects d'implémentation

Il nous a fallut environ six semaines pour réaliser un DSA "*multi-thread*" à partir du DSA bi-processus.

Nous avons ensuite passé deux semaines à tester les performances du DSA "*multi-thread*" et celles d'autres versions du DSA.

9.9 Conclusions

Nous avons fait percevoir dans ce chapitre la difficulté pratique de mettre en oeuvre le "*multi-thread*" dans une application existante.

Ensuite, point peut-être le plus important, nous avons évalué les performances d'un DSA "*multi-thread*" pour savoir si notre travail avait servi à quelque chose. On s'est en fait rendu compte qu'il y avait une amélioration des temps de réponse mais que celle-ci n'était pas toujours très importante.

Pour que les temps de réponse soient nettement inférieurs à ceux que nous avons constaté, nous mettons en avance plusieurs causes qui permettraient, peut-être, d'améliorer encore les performances du DSA "*multi-thread*" mono-processus si elles étaient éliminées :

- les tâches sont trop dépendantes les unes des autres. En effet, dans notre cas, à part lire dans la BD et transférer les résultats, les "*threads*" ne font pas grand chose. Les CPU sont donc sous-utilisés et des goulots d'étranglement peuvent apparaître plus facilement;
- dans le code source du DSA, il y a beaucoup d'appels aux fonctions générées par MAVROS. Or, ces fonctions MAVROS ne sont pas bien adaptées à une utilisation dans un environnement "*multi-thread*" : les structures de données ressemblant à des "poupées russes", les fonctions qui leur sont associées s'appellent énormément entre elles. Ainsi, un appel à `dsk_dist_name_free()` va provoquer plusieurs appels à la fonction `dsk_rdname_free()` qui va elle-même appeler plusieurs fois `dsk_assertion_free()`. Or, dans un environnement "*multi-thread*", les `free()`, appelés par ces fonctions, sont sérialisés (vu qu'ils manipulent une liste, en mémoire globale, de blocs mémoire libres). Par conséquent, les `free()` appelés tant de fois réduisent de façon substantielle le parallélisme des applications "*multi-thread*". Une solution à ce problème serait que MAVROS gère un "*buffer*" mémoire alloué une fois pour toute avec une taille suffisante pour éviter de devoir, au coup par coup, allouer et "désallouer" de la mémoire;
- lorsque des ressources sont verrouillées et que plusieurs "*threads*" sont en attente, le système ne garantit aucun ordre d'acquisition du verrou. Des délais seraient peut-être dus à ce phénomène. Certains "*threads*" peuvent, en effet, être retardés plusieurs fois au profit d'autres "*threads*".

Précisons toutefois que, malheureusement, nous n'avons pas eu l'opportunité de tester ces hypothèses.

10. Conclusion

En guise de conclusion, nous allons brièvement remettre en lumière les principaux sujets qui ont jalonné ce mémoire. Et, pour terminer, nous parlerons d'un prolongement possible de ce mémoire.

Le service d'annuaire X.500 a été conçu dès le départ pour être offert tant à des utilisateurs humains qu'à des applications informatiques.

Des implémentations de ce service OSI sont, à l'heure actuelle, déjà largement répandues. Elles offrent un service bien plus riche que le plus proche de leurs concurrents du monde Internet (en l'occurrence, DNS).

Bien que nous n'avons fait qu'effleurer ce sujet, nous pouvons informer le lecteur du fait que les mécanismes d'authentification des utilisateurs et de contrôle d'accès sont développés en profondeur dans la version '93 du standard. X.500 se défend donc bien dans ce domaine à la mode.

Après avoir fait le tour des questions principales au sujet de l'annuaire X.500, nous nous sommes quelque peu attardés sur l'étude détaillée d'une implémentation de l'annuaire : l'implémentation Forum Lookup développée par la société française Telis.

L'intérêt de cette étude s'est porté sur deux domaines plus particuliers : le protocole SOLO et la programmation "*multi-thread*".

Nous avons montré l'intérêt du protocole SOLO en attirant l'attention sur son caractère utilitaire pour un utilisateur humain. L'adoption par ce protocole d'une syntaxe conviviale de nommage des entrées comble, en effet, une lacune certaine du protocole DAP. Pour cette raison, nous espérons que d'autres implémentations de l'annuaire opteront pour ce protocole dans un futur proche.

Le dernier projet que nous avons dû réaliser lors de notre stage (après SOLO-Ping et DB-Audit) a de loin été le plus enrichissant. Il a constitué une mise en pratique des concepts de la programmation "*multi-thread*" dans le cadre de Forum Lookup. Après avoir expliqué les différentes évolutions qui ont été apportées au DSA de Forum Lookup, les tests de performance que nous avons effectué sur notre version "*multi-thread*" du DSA nous ont permis de montrer la pertinence d'une telle implémentation. D'ailleurs, aujourd'hui, nombreuses sont les applications qui suivent ce modèle de programmation.

En ce qui concerne l'architecture du DSA de Forum Lookup, nous allons maintenant proposer une autre piste qui aurait mérité d'être ouverte mais que les limites de ce texte nous ont forcé à délaisser : il s'agit des E/S asynchrones. Tout comme la programmation "*multi-thread*", la programmation qui se base sur les E/S asynchrones permet de traiter les requêtes des utilisateurs de manière asynchrone. Bien que nous n'avons aucune connaissance particulière sur ce style de programmation (et, en particulier, sur ses performances), nous pensons que l'exploration de cette voie pourrait constituer un excellent prolongement à ce mémoire.

Annexes

Annexe A : classes d'objets et types d'attribut

Nous décrivons ici les classes d'objets de la version '88 du standard qui sont les plus utiles. Ces descriptions en ASN.1 sont accompagnées des descriptions de types d'attribut et des longueurs maximales de leurs valeurs.

Classes d'objets standards

```
top OBJECT-CLASS
    MUST CONTAIN {
        objectClass}
::= {objectClass 0}


alias OBJECT-CLASS
    SUBCLASS OF top
    MUST CONTAIN {
        aliasedObjectName}
::= {objectClass 1}


country OBJECT-CLASS
    SUBCLASS OF top
    MUST CONTAIN {
        countryName}
    MAY CONTAIN {
        description,
        searchGuide}
::= {objectClass 2}


locality OBJECT-CLASS
    SUBCLASS OF top
    MAY CONTAIN {
        description,
        localityName,
        stateOrProvinceName,
        searchGuide,
        seeAlso,
        streetAddress}
::= {objectClass 3}


organization OBJECT-CLASS
    SUBCLASS OF top
    MUST CONTAIN {
        organizationName}
    MAY CONTAIN {
        organizationalAttributeSet}
::= {objectClass 4}


organizationalUnit OBJECT-CLASS
    SUBCLASS OF top
    MUST CONTAIN {
        organizationalUnitName}
    MAY CONTAIN {
        organizationalAttributeSet}
::= {objectClass 5}
```

```
person OBJECT-CLASS
    SUBCLASS OF top
    MUST CONTAIN {
        commonName,
        surname}
    MAY CONTAIN {
        description,
        seeAlso,
        telephoneNumber,
        userPassword}
::= {objectClass 6}


organizationalPerson OBJECT-CLASS
    SUBCLASS OF person
    MAY CONTAIN {
        localeAttributeSet,
        organizationalUnitName,
        postalAttributeSet,
        telecommunicationAttributeSet,
        title}
::= {objectClass 7}


organizationalRole OBJECT-CLASS
    SUBCLASS OF top
    MUST CONTAIN {
        commonName}
    MAY CONTAIN {
        description,
        localeAttributeSet,
        organizationalUnitName,
        postalAttributeSet,
        preferredDeliveryMethod,
        roleOccupant,
        seeAlso,
        telecommunicationAttributeSet}
::= {objectClass 8}


applicationEntity OBJECT-CLASS
    SUBCLASS OF top
    MUST CONTAIN {
        commonName,
        presentationAddress}
    MAY CONTAIN {
        description,
        localityName,
        organizationName,
        organizationalUnitName,
        seeAlso,
        supportedApplicationContext}
::= {objectClass 12}


dSA OBJECT-CLASS
    SUBCLASS OF applicationEntity
    MAY CONTAIN {
        knowledgeInformation}
::= {objectClass 13}
```

Types d'attribut standards

```
objectClass ObjectClass
    ::= {attributeType 0}
```

```
aliasedObjectName AliasedObjectName
    ::= {attributeType 1}


knowledgeInformation ATTRIBUTE
    WITH ATTRIBUTE-SYNTAX caseIgnoreString
    ::= {attributeType 2}


commonName ATTRIBUTE
    WITH ATTRIBUTE-SYNTAX caseIgnoreStringSyntax
    (SIZE (1..ub-common-name))
    ::= {attributeType 3}


surname ATTRIBUTE
    WITH ATTRIBUTE-SYNTAX caseIgnoreStringSyntax
    (SIZE (1..ub-surname))
    ::= {attributeType 4}


countryName ATTRIBUTE
    WITH ATTRIBUTE-SYNTAX PrintableString
    (SIZE (1..ub-country-code))
    SINGLE VALUE
    ::= {attributeType 6}


localityName ATTRIBUTE
    WITH ATTRIBUTE-SYNTAX caseIgnoreStringSyntax
    (SIZE (1..ub-locality-name))
    ::= {attributeType 7}


stateOrProvinceName ATTRIBUTE
    WITH ATTRIBUTE-SYNTAX caseIgnoreStringSyntax
    (SIZE (1..ub-state-name))
    ::= {attributeType 8}


streetAddress ATTRIBUTE
    WITH ATTRIBUTE-SYNTAX caseIgnoreStringSyntax
    (SIZE (1..ub-street-address))
    ::= {attributeType 9}


organizationName ATTRIBUTE
    WITH ATTRIBUTE-SYNTAX caseIgnoreStringSyntax
    (SIZE (1..ub-organization-name))
    ::= {attributeType 10}


organizationalUnitName ATTRIBUTE
    WITH ATTRIBUTE-SYNTAX caseIgnoreStringSyntax
    (SIZE (1..ub-organizational-unit-name))
    ::= {attributeType 11}


title ATTRIBUTE
    WITH ATTRIBUTE-SYNTAX caseIgnoreStringSyntax
    (SIZE (1..ub-title))
    ::= {attributeType 12}
```

```
description ATTRIBUTE
    WITH ATTRIBUTE-SYNTAX caseIgnoreStringSyntax
    (SIZE (1..ub-description))
    ::= {attributeType 13}


searchGuide ATTRIBUTE
    WITH ATTRIBUTE-SYNTAX Guide
    ::= {attributeType 14}


telephoneNumber ATTRIBUTE
    WITH ATTRIBUTE-SYNTAX telephoneNumberSyntax
    (SIZE (1..ub-telephone-number))
    ::= {attributeType 20}


preferredDeliveryMethod ATTRIBUTE
    WITH ATTRIBUTE-SYNTAX deliveryMethod
    ::= {attributeType 28}


presentationAddress ATTRIBUTE
    WITH ATTRIBUTE-SYNTAX PresentationAddress
    MATCHES FOR EQUALITY
    ::= {attributeType 29}


supportedApplicationContext ATTRIBUTE
    WITH ATTRIBUTE-SYNTAX objectIdentifierSyntax
    ::= {attributeType 30}


roleOccupant ATTRIBUTE
    WITH ATTRIBUTE-SYNTAX distinguishedNameSyntax
    ::= {attributeType 33}


seeAlso ATTRIBUTE
    WITH ATTRIBUTE-SYNTAX distinguishedNameSyntax
    ::= {attributeType 34}


userPassword ATTRIBUTE
    WITH ATTRIBUTE-SYNTAX Userpassword
    ::= {attributeType 35}
```

Bornes supérieures de la longueur des valeurs d'attribut

```
ub-common-name INTEGER ::= 64
ub-surname INTEGER ::= 64
ub-country-code INTEGER ::= 4
ub-locality-name INTEGER ::= 128
ub-state-name INTEGER ::= 128
ub-street-address INTEGER ::= 128
ub-organization-name INTEGER ::= 64
ub-organizational-unit-name INTEGER ::= 64
ub-title INTEGER ::= 64
ub-description INTEGER ::= 1024
ub-telephone-number INTEGER ::= 32
```

Annexe B : tests de performance du DSA multi-thread

Cet annexe rassemble, sous forme numérique et sous forme graphique, les résultats les plus intéressants que nous avons glané de nos nombreux tests de performance des différentes versions du DSA de Forum Lookup.

Avant de donner des informations utiles pour la lecture de ces neuf feuilles de calcul, précisons que la BD que nous avons employée lors de nos tests contenait 27.000 entrées. Le DIT qu'elles formaient avait aussi pour particularité d'avoir six niveaux d'unités organisationnelles, comme le montre la Figure 47. Cela permettait de tester les performances du DSA dans des conditions réelles d'utilisation, voire même extrêmes.

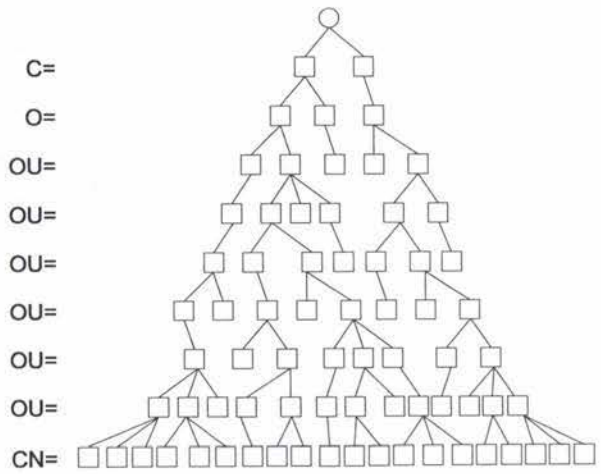


Figure 47 - Organisation hiérarchique des entrées de la base de test

Les sept premières feuilles de calcul présentent les temps de réponse des différents DSA lorsqu'ils sont soumis aux 210 requêtes d'un client isolé.

La huitième feuille, elle, résume les sept premières feuilles de calcul en ne reprenant que les temps moyens de réponse à chaque type de requête dans le cas d'un client isolé.

Enfin, la neuvième et dernière feuille de calcul agrège tous les temps de réponse des DSA lorsque quatre clients les interrogent simultanément.

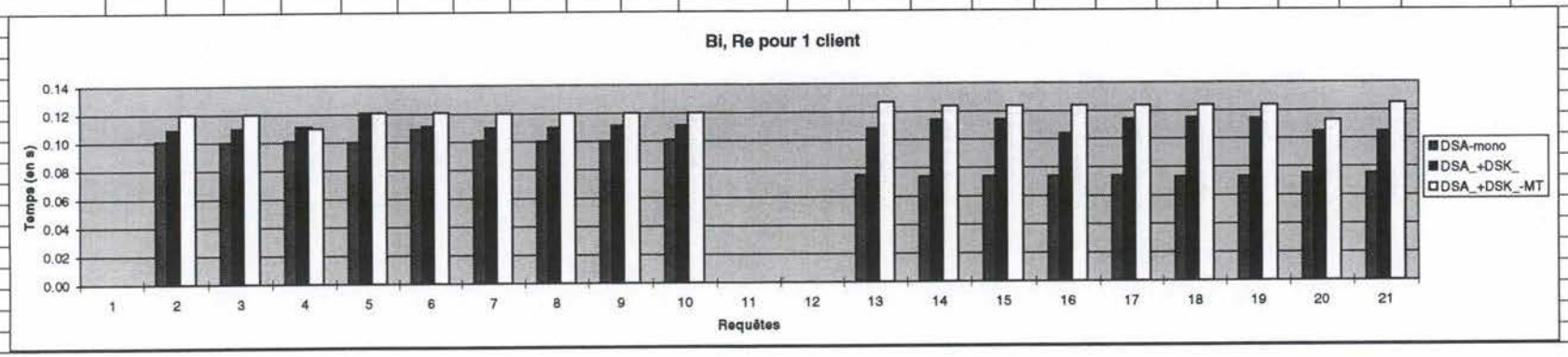
Pour comprendre les feuilles qui suivent, ils nous reste finalement à expliquer les abréviations utilisées :

- "`Bi`" signifie *Bind*;
- "`Re`" signifie *Read*;
- "`Se2`" signifie *Search* dont la portée de recherche s'étend à tout le sous-arbre commençant à l'entrée de base;
- "`Se1`" signifie *Search* dont la portée de recherche se limite aux entrées directement subordonnées à l'entrée de base;
- "`=`" signifie que la recherche se fait par égalité;

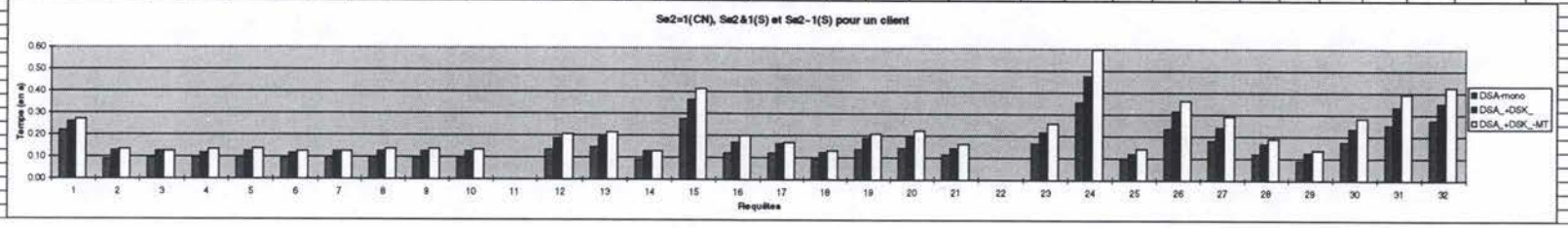
- "&" signifie que la recherche se fait par sous-chaîne;
- "~" signifie que la recherche se fait par approximation;
- "5" signifie que l'entrée de base pour la recherche est située au cinquième niveau d'unité organisationnelle;
- "6" signifie que l'entrée de base pour la recherche est située au sixième niveau d'unité organisationnelle;
- les attributs notés entre parenthèses indiquent les attributs sur lesquels portent la recherche.

Ainsi, par exemple, "Se2&1(S)" signifie recherche par sous-chaîne dans tout le sous-arbre commençant à l'entrée de base (située au premier niveau d'unités organisationnelles) sur l'attribut "S".

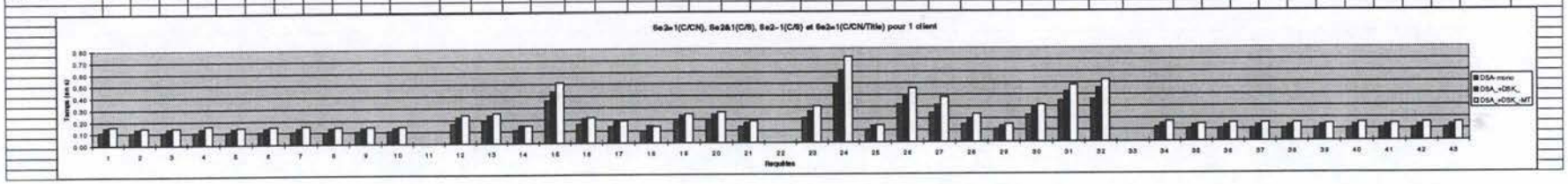
1 client, base de 27.000 entrées																							
	Bi											Re											
Temps utilisateur (en s)																							
DSA-mono	x0,891	0.10	0.10	0.10	0.10	0.11	0.10	0.10	0.10	0.10		x0,552	0.08	0.07	0.07	0.07	0.07	0.07	0.07	0.08	0.08		
DSA_+DSK_	x0,851	0.11	0.11	0.11	0.12	0.11	0.11	0.11	0.11	0.11		x0,539	0.11	0.11	0.11	0.10	0.11	0.12	0.11	0.11	0.11		
DSA_+DSK_-MT	x0,838	0.12	0.12	0.11	0.12	0.12	0.12	0.12	0.12	0.12		x0,568	0.13	0.12	0.12	0.12	0.12	0.12	0.12	0.11	0.13		
Différence (en ms)											**Moyenne**												**Moyenne**
bi / mono	#####	8	10	10	21	2	9	10	11	10	10	#####	33	40	40	30	40	42	41	30	30	36	
MT / bi	#####	11	10	-1	0	10	10	10	9	9	8	#####	19	10	10	20	10	9	10	8	20	13	



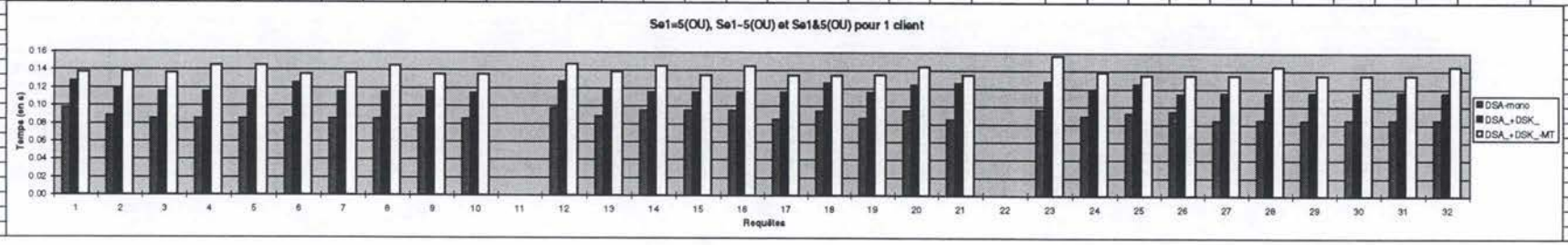
1 client, base de 27.000 entrées																																		
	Se2=1, CN											Se2&1, S											Se2-1, S											
Temps utilisateur (en s)																																		
DSA-mono	0.22	0.09	0.10	0.10	0.10	0.10	0.09	0.10	0.10	0.09		0.14	0.15	0.09	0.27	0.12	0.12	0.09	0.14	0.14	0.11		0.17	0.35	0.10	0.23	0.18	0.12	0.09	0.17	0.25	0.27		
DSA_+DSK_	0.26	0.13	0.13	0.12	0.13	0.12	0.13	0.13	0.13	0.13		0.19	0.20	0.13	0.36	0.17	0.16	0.12	0.19	0.19	0.14		0.22	0.48	0.12	0.32	0.24	0.17	0.13	0.24	0.33	0.35		
DSA_+DSK_-MT	0.27	0.13	0.13	0.14	0.14	0.13	0.13	0.14	0.14	0.14		0.21	0.22	0.13	0.42	0.20	0.17	0.13	0.21	0.22	0.16		0.26	0.60	0.14	0.36	0.29	0.19	0.14	0.28	0.39	0.43		
Différence (en ms)										Moyenne												Moyenne											Moyenne
bi / mono	39	39	30	22	31	21	32	29	32	32	31	51	49	40	90	48	43	30	50	49	29	48	50	121	24	82	59	46	39	61	81	79	64	
MT / bi	12	5	1	18	10	10	0	10	10	9	9	20	20	1	51	31	8	10	20	30	20	21	39	121	23	49	51	23	10	48	60	73	50	



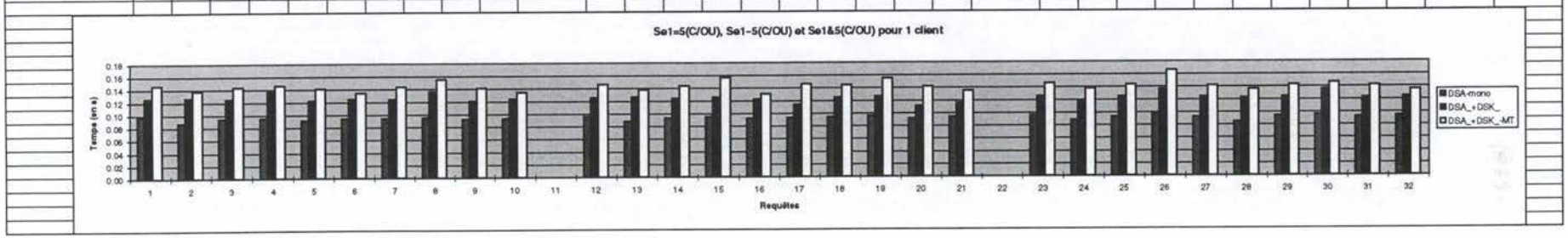
1 client, base de 27.000 entrées	Se2e1, C/CN											Se2&1, C/S											Se2-1, C/S											Se2e1, C/CN/Title										
Temps utilisateur (en s)																																													
DSA-mono	0.11	0.11	0.11	0.11	0.11	0.11	0.11	0.10	0.11	0.11		0.17	0.19	0.11	0.36	0.16	0.14	0.10	0.18	0.19	0.13		0.21	0.50	0.11	0.39	0.24	0.15	0.10	0.23	0.35	0.36		0.12	0.11	0.11	0.11	0.11	0.11	0.11	0.11	0.11	0.11	
DSA_+DSK_	0.15	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.14		0.22	0.24	0.15	0.44	0.21	0.20	0.14	0.23	0.24	0.17		0.26	0.61	0.14	0.36	0.31	0.20	0.14	0.26	0.42	0.45		0.15	0.14	0.14	0.14	0.14	0.13	0.14	0.14	0.14	0.14	
DSA_+DSK_MT	0.16	0.15	0.15	0.16	0.14	0.15	0.15	0.15	0.15	0.15		0.24	0.26	0.15	0.52	0.22	0.19	0.14	0.25	0.26	0.18		0.31	0.75	0.15	0.46	0.36	0.24	0.15	0.32	0.48	0.53		0.17	0.15	0.16	0.16	0.16	0.15	0.16	0.15	0.16	0.16	
Différence (en ms)											Moyenne											Moyenne											Moyenne										Moyenne
bi / mono	41	28	30	30	30	30	31	39	30	29	32	50	50	39	74	47	54	42	50	47	36	49	57	116	36	76	67	46	30	57	72	87	66	30	29	30	31	30	27	30	31	31	30	30
MT / bi	10	10	9	21	8	11	17	10	10	9	12	20	21	1	78	11	-6	-1	19	21	11	18	43	116	3	63	66	36	10	32	50	72	51	16	10	20	19	16	12	19	9	19	20	16



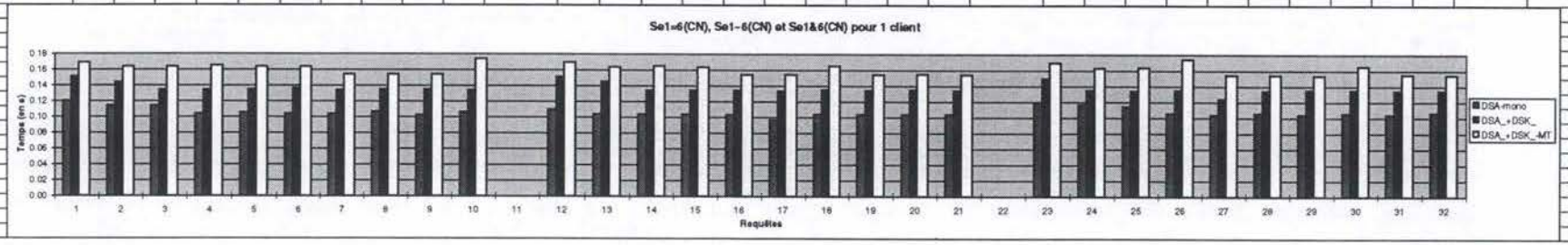
1 client, base de 27.000 entrées																																	
	Se1=5, OU											Se1-5, OU											Se1&5, OU										
Temps utilisateur (en s)																																	
DSA-mono	0.10	0.09	0.09	0.09	0.09	0.09	0.09	0.09	0.09	0.09		0.10	0.09	0.10	0.10	0.10	0.09	0.09	0.09	0.10	0.09		0.10	0.09	0.09	0.09	0.08	0.09	0.08	0.09	0.09	0.09	
DSA_+DSK_	0.13	0.12	0.12	0.12	0.12	0.13	0.12	0.12	0.12	0.11		0.13	0.12	0.12	0.12	0.12	0.12	0.13	0.12	0.12	0.13		0.13	0.12	0.13	0.11	0.12	0.12	0.12	0.12	0.12	0.12	
DSA_+DSK_-MT	0.14	0.14	0.14	0.15	0.15	0.14	0.14	0.15	0.14	0.14		0.15	0.14	0.15	0.13	0.15	0.13	0.13	0.14	0.14	0.14		0.16	0.14	0.14	0.14	0.14	0.15	0.14	0.14	0.14	0.15	
Différence (en ms)											Moyenne											Moyenne											Moyenne
bi / mono	30	31	30	30	31	40	30	30	31	29	31	30	31	20	20	20	30	32	29	29	41	28	32	29	33	20	31	30	31	30	31	30	30
MT / bi	10	19	21	30	29	10	21	30	19	21	21	20	19	30	19	30	19	8	20	20	9	19	29	20	10	21	20	30	20	20	19	30	22



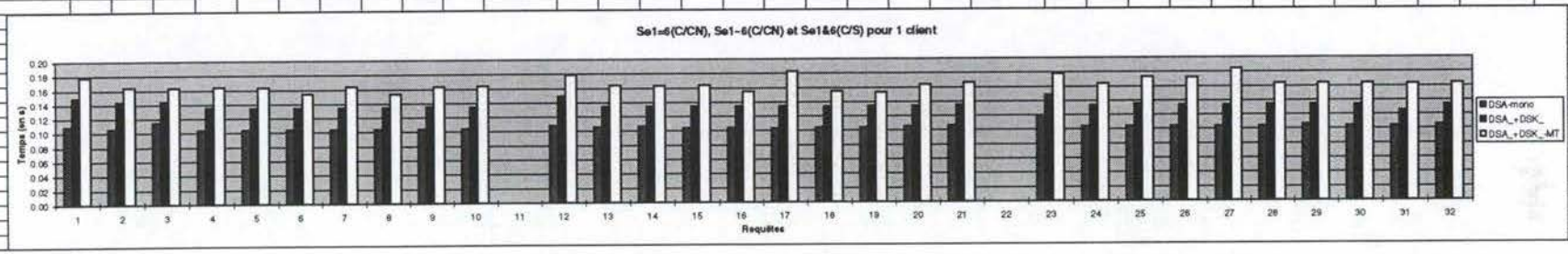
1 client, base de 27.000 entrées																																		
	Se1=5, C/OU											Se1-5, C/OU											Se1&5, C/OU											
Temps utilisateur (en s)																																		
DSA-mono	0.10	0.09	0.09	0.10	0.09	0.09	0.09	0.10	0.09	0.09		0.10	0.09	0.09	0.10	0.09	0.09	0.09	0.10	0.09	0.09		0.10	0.09	0.09	0.10	0.09	0.08	0.09	0.10	0.09	0.09		
DSA_+DSK_	0.13	0.13	0.13	0.14	0.12	0.13	0.12	0.14	0.12	0.12		0.13	0.13	0.12	0.13	0.12	0.11	0.13	0.13	0.11	0.12		0.13	0.12	0.13	0.14	0.12	0.12	0.12	0.14	0.12	0.12		
DSA_+DSK_-MT	0.15	0.14	0.14	0.15	0.14	0.13	0.14	0.16	0.14	0.13		0.15	0.14	0.14	0.16	0.13	0.15	0.15	0.16	0.14	0.14		0.15	0.14	0.14	0.17	0.14	0.14	0.14	0.15	0.14	0.14		
Différence (en ms)											Moyenne											Moyenne											Moyenne
bi / mono	29	40	31	42	31	31	30	41	29	31	34	30	39	31	31	30	21	31	31	20	21	29	30	29	32	39	32	38	30	40	30	30	33	
MT / bi	20	11	19	9	19	9	20	19	20	10	16	20	11	20	31	9	32	20	28	31	20	22	20	21	19	30	18	14	19	10	20	11	18	



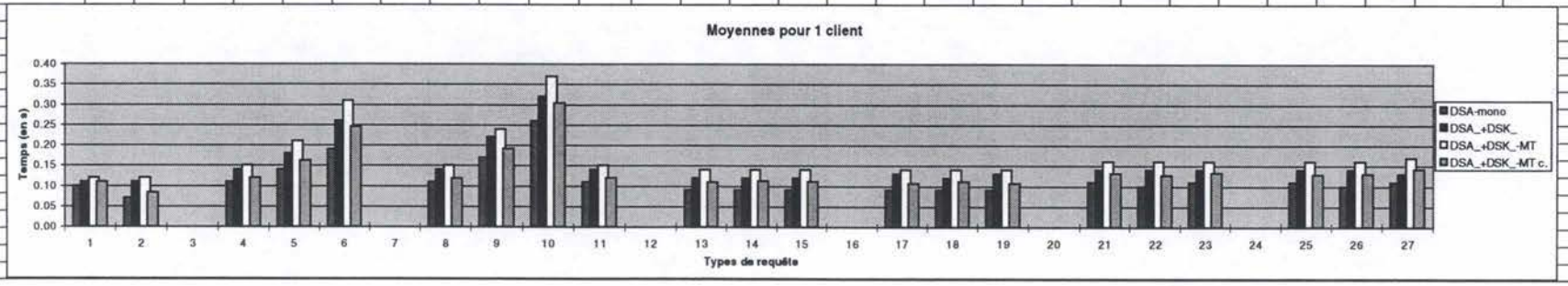
1 client, base de 27.000 entrées																																	
	Se1=6, CN											Se1-6, CN											Se1&6, CN										
Temps utilisateur (en s)																																	
DSA-mono	0.12	0.11	0.11	0.10	0.11	0.10	0.10	0.11	0.10	0.11		0.11	0.10	0.10	0.10	0.10	0.10	0.10	0.10	0.10	0.10		0.12	0.12	0.11	0.11	0.10	0.11	0.10	0.11	0.10	0.11	
DSA_+DSK_	0.15	0.14	0.13	0.13	0.14	0.14	0.13	0.14	0.14	0.13		0.15	0.15	0.13	0.13	0.13	0.13	0.14	0.13	0.14	0.13		0.15	0.14	0.13	0.14	0.12	0.13	0.14	0.14	0.13	0.13	
DSA_+DSK_-MT	0.17	0.16	0.16	0.17	0.16	0.16	0.15	0.15	0.15	0.17		0.17	0.16	0.17	0.16	0.15	0.15	0.17	0.15	0.16	0.15		0.17	0.16	0.16	0.17	0.15	0.15	0.15	0.17	0.16	0.15	
Différence (en ms)											Moyenne											Moyenne											Moyenne
bi / mono	31	30	20	30	30	32	30	28	33	28	29	41	41	30	30	31	37	31	30	31	30	33	31	19	20	30	20	29	31	30	30	28	27
MT / bi	18	20	30	31	29	28	20	19	19	40	25	19	19	31	30	20	21	30	20	20	20	23	20	28	30	39	30	20	18	30	21	20	26



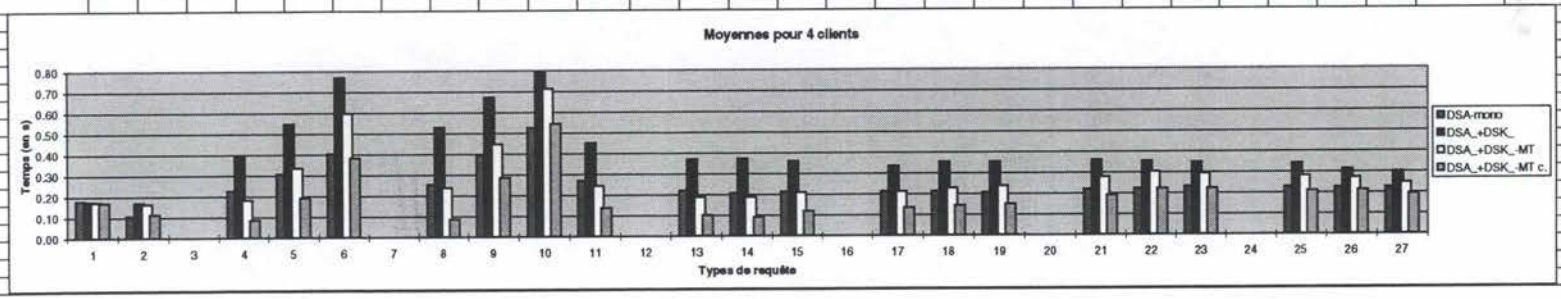
1 client, base de 27.000 entrées																																	
	Se1=6, C/CN											Se1-6, C/CN												Se1&6, C/S										
Temps utilisateur (en s)																																	
DSA-mono	0.11	0.11	0.11	0.10	0.10	0.10	0.10	0.10	0.10	0.10		0.11	0.11	0.11	0.10	0.10	0.10	0.10	0.10	0.11	0.11		0.12	0.10	0.10	0.10	0.10	0.10	0.11	0.10	0.10	0.11	
DSA_+DSK_	0.15	0.14	0.14	0.14	0.13	0.13	0.13	0.13	0.14	0.13		0.15	0.13	0.13	0.13	0.13	0.13	0.13	0.13	0.13	0.14		0.15	0.13	0.14	0.13	0.13	0.13	0.13	0.13	0.13	0.13	
DSA_+DSK_-MT	0.18	0.16	0.16	0.16	0.16	0.15	0.16	0.15	0.16	0.16		0.18	0.16	0.16	0.16	0.16	0.18	0.16	0.15	0.16	0.17		0.18	0.16	0.17	0.17	0.18	0.16	0.16	0.16	0.16	0.16	
Différence (en ms)											Moyenne											Moyenne											Moyenne
bi / mono	40	38	30	31	30	30	30	30	31	30	32	41	29	28	30	30	31	29	30	29	29	31	29	29	31	29	29	30	28	29	21	28	28
MT / bi	29	21	19	29	29	20	30	19	28	30	25	30	30	29	30	21	49	22	19	30	31	29	30	31	38	39	51	30	30	31	38	31	35



1 client, base de 27.000 entrées																													
	Bi	Re		Se2=1	Se2&1	Se2-1		Se2=1	Se2&1	Se2-1	Se2=1		Se1=5	Se1-5	Se1&5		Se1=5	Se1-5	Se1&5		Se1=6	Se1-6	Se1&6		Se1=6	Se1-6	Se1&6		
				CN	S	S		C/CN	C/S	C/S	C/CN/T		OU	OU	OU		C/OU	C/OU	C/OU		CN	CN	CN		C/CN	C/CN	C/S		
Temps utilisateur moyen (en s)																													
DSA-mono	0.10	0.07		0.11	0.14	0.19		0.11	0.17	0.26	0.11		0.09	0.09	0.09		0.09	0.09	0.09		0.11	0.10	0.11		0.11	0.10	0.11		
DSA_+DSK_	0.11	0.11		0.14	0.18	0.26		0.14	0.22	0.32	0.14		0.12	0.12	0.12		0.13	0.12	0.13		0.14	0.14	0.14		0.14	0.14	0.13		
DSA_+DSK_-MT	0.12	0.12		0.15	0.21	0.31		0.15	0.24	0.37	0.15		0.14	0.14	0.14		0.14	0.14	0.14		0.16	0.16	0.16		0.16	0.16	0.17		
Différence moyenne (en ms)																													
bi / mono	10	36		31	48	64		32	49	66	30		31	28	30		34	29	33		29	33	27		32	31	28		
MT / bi	8	13		9	21	50		12	18	51	16		21	19	22		16	22	18		25	23	26		25	29	35		
Temps utilisateur corrigé (en s)																													
DSA_+DSK_-MT c.	0.11	0.08		0.12	0.16	0.25		0.12	0.19	0.30	0.12		0.11	0.11	0.11		0.11	0.11	0.11		0.13	0.13	0.13		0.13	0.13	0.14		



4 clients, base de 27.000 entrées																												
		Bi	Re		Se2=1	Se2&1	Se2-1		Se2=1	Se2&1	Se2-1	Se2=1		Se1=5	Se1-5	Se1&5		Se1=5	Se1-5	Se1&5		Se1=6	Se1-6	Se1&6		Se1=6	Se1-6	Se1&6
					CN	S	S		C/CN	C/S	C/S	C/CN/T		OU	OU	OU		C/OU	C/OU	C/OU		CN	CN	CN		C/CN	C/CN	C/S
Temps utilisateur moyen (en s)																												
DSA-mono	client 1	0.190	0.123		0.150	0.242	0.347		0.187	0.325	0.627	0.296		0.177	0.152	0.173		0.196	0.193	0.170		0.181	0.185	0.219		0.207	0.196	0.220
	client 2	0.174	0.115		0.251	0.314	0.449		0.260	0.429	0.505	0.270		0.252	0.220	0.227		0.216	0.211	0.217		0.229	0.239	0.236		0.244	0.232	0.231
	client 3	0.177	0.094		0.250	0.321	0.420		0.274	0.415	0.505	0.275		0.228	0.227	0.224		0.216	0.219	0.216		0.225	0.236	0.230		0.232	0.240	0.223
	client 4	0.174	0.097		0.255	0.353	0.402		0.288	0.408	0.462	0.235		0.220	0.221	0.225		0.217	0.226	0.216		0.236	0.228	0.237		0.234	0.230	0.232
Moyenne des 4 clients	DSA-mono	**0.18**	**0.11**		**0.23**	**0.31**	**0.40**		**0.25**	**0.39**	**0.52**	**0.27**		**0.22**	**0.21**	**0.21**		**0.21**	**0.21**	**0.20**		**0.22**	**0.22**	**0.23**		**0.23**	**0.22**	**0.23**
DSA_+DSK_	client 1	0.194	0.185		0.224	0.737	0.979		0.888	0.929	0.794	0.403		0.338	0.380	0.369		0.376	0.389	0.376		0.364	0.330	0.277		0.275	0.162	0.153
	client 2	0.198	0.181		0.490	0.480	0.673		0.417	0.611	0.708	0.478		0.366	0.375	0.353		0.330	0.328	0.341		0.364	0.353	0.373		0.364	0.355	0.350
	client 3	0.169	0.179		0.427	0.503	0.686		0.404	0.501	0.831	0.500		0.333	0.367	0.347		0.313	0.324	0.327		0.345	0.348	0.367		0.361	0.374	0.356
	client 4	0.141	0.135		0.409	0.465	0.742		0.401	0.642	0.843	0.424		0.437	0.361	0.345		0.321	0.371	0.366		0.361	0.385	0.376		0.363	0.358	0.344
Moyenne des 4 clients	DSA_+DSK_	**0.18**	**0.17**		**0.39**	**0.55**	**0.77**		**0.53**	**0.67**	**0.79**	**0.45**		**0.37**	**0.37**	**0.36**		**0.34**	**0.35**	**0.35**		**0.36**	**0.35**	**0.35**		**0.34**	**0.31**	**0.30**
DSA_+DSK_-MT	client 1	0.188	0.172		0.174	0.315	0.626		0.222	0.420	0.757	0.268		0.193	0.191	0.197		0.217	0.232	0.232		0.288	0.314	0.304		0.293	0.275	0.252
	client 2	0.170	0.185		0.189	0.335	0.613		0.241	0.425	0.707	0.229		0.169	0.177	0.209		0.202	0.221	0.221		0.264	0.299	0.286		0.292	0.286	0.272
	client 3	0.172	0.147		0.161	0.317	0.613		0.207	0.403	0.746	0.271		0.203	0.191	0.209		0.216	0.228	0.244		0.280	0.306	0.308		0.279	0.280	0.237
	client 4	0.163	0.141		0.202	0.371	0.537		0.280	0.540	0.641	0.202		0.190	0.197	0.217		0.217	0.229	0.259		0.276	0.299	0.273		0.264	0.243	0.225
Moyenne des 4 clients	DSA_+DSK_-MT	**0.17**	**0.16**		**0.18**	**0.33**	**0.60**		**0.24**	**0.45**	**0.71**	**0.24**		**0.19**	**0.19**	**0.21**		**0.21**	**0.23**	**0.24**		**0.28**	**0.30**	**0.29**		**0.28**	**0.27**	**0.25**
Différence moyenne (en ms)																												
bi / mono (1 client)		10	36		31	48	64		32	49	66	30		31	28	30		34	29	33		29	33	27		32	31	28
bi / mono (4 clients)		-3	63		161	239	366		275	277	269	182		149	166	146		124	141	148		141	132	118		112	88	74
bi / mono (moyenne 1 et 4 clients)		3	49		96	143	215		154	163	168	106		90	97	88		79	85	90		85	83	72		72	59	51
MT / bi		-2	-9		-206	-212	-173		-290	-224	-81	-209		-180	-182	-151		-122	-126	-114		-82	-50	-56		-59	-41	-54
Temps utilisateur corrigé (en s)																												
Moyenne des 4 clients	DSA_+DSK_-MT c.	**0.17**	**0.11**		**0.09**	**0.19**	**0.38**		**0.08**	**0.28**	**0.55**	**0.14**		**0.10**	**0.09**	**0.12**		**0.13**	**0.14**	**0.15**		**0.19**	**0.22**	**0.22**		**0.21**	**0.21**	**0.20**



Acronymes

ACDF	Access Control Decision Function
ACSE	Association Control Service Element
ASCII	American Standard Code for Information Interchange
ASN.1	Abstract Syntax Notation One
AVA	Attribute Value Assertion
BD	Base de Données
BER	Basic Encoding Rules
C	Country name
CCITT	Comité Consultatif International pour le Télégraphe et le Téléphone
CN	Common Name
CPU	Central Processing Unit
DAP	Directory Access Protocol
DIB	Directory Information Base
DISP	Directory Information Shadowing Protocol
DIT	Directory Information Tree
DMD	Directory Management Domain
DN	Distinguished Name
DNS	Domain Name System
DOP	Directory Operational binding management Protocol
DSA	Directory System Agent
DSP	Directory System Protocol
DUA	Directory User Agent
EDI	Electronic Data Interchange
E/S	Entrée/Sortie
HTML	HyperText Markup Language
IHM	Interface Homme-Machine
IP	Internet Protocol
ISO	International Standards Organization
ITU	International Telecommunication Union
L	Locality name
LWP	LightWeight Process

MT	Multi-Thread
O	Organization name
OID	Object IDentifier
OSI	Open Systems Interconnection
OU	Organizational Unit name
PAD	Packet Assembler Disassembler
PAVI	Point d'Accès VIdéotex
PDU	Protocol Data Unit
POSIX	Portable Operating System for unIX
PSAP	Presentation Service Access Point
R&D	Recherche & Développement
RDN	Relative Distinguished Name
RFC	Request For Comments
RLE	Réseau Local d'Entreprise
ROSE	Remote Operations Service Element
RPC	Remote Procedure Call
RTC	Réseau Téléphonique Commuté
S	Surname
S&C	Systèmes & Communications
SOLO	Simple Object LOokup
SQL	Structured Query Language
SSII	Société de Service et d'Ingénierie Informatique
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
UFN	User-Friendly Naming
UI	Unix International
URL	Universal Resource Locator
WAN	Wide Area Network
WWW	World-Wide Web

Références bibliographiques

[BRUN-92] M. BRUNET,
INTRODUCTION A X.500 : PRINCIPES ET DEFINITIONS ABSTRAITES,
Mémoire de fin d'études,
FUNDP - Institut d'Informatique, juin 1992.

[CHAD-94] D. CHADWICK,
UNDERSTANDING X.500 - THE DIRECTORY,
Chapman & Hall, London, 1994.

[FORU-95a] EQUIPE FORUM LOOKUP,
ADMINISTRATION DSA - FORUM LOOKUP V1.1,
TELIS Systèmes & Communications, 8 août 1995.

[FORU-95b] EQUIPE FORUM LOOKUP,
ADMINISTRATION DUA-SERVEUR - FORUM LOOKUP V1.1,
TELIS Systèmes & Communications, 12 septembre 1995.

[FORU-95c] EQUIPE FORUM LOOKUP,
ADMINISTRATION SOLO-ROUTEUR - FORUM LOOKUP V1.1,
TELIS Systèmes & Communications, 12 septembre 1995.

[FORU-95d] EQUIPE FORUM LOOKUP,
ADMINISTRATION SOLO-SERVEUR - FORUM LOOKUP V1.1,
TELIS Systèmes & Communications, 12 septembre 1995.

[FORU-95e] EQUIPE FORUM LOOKUP,
ANNUAIRE DE SITE - FORUM LOOKUP V1.1,
TELIS Systèmes & Communications, 12 septembre 1995.

[FORU-95f] EQUIPE FORUM LOOKUP,
INSTALLATION PLATE-FORME - FORUM LOOKUP V1.1,
TELIS Systèmes & Communications, 13 septembre 1995.

[FORU-95g] EQUIPE FORUM LOOKUP,
"L'ANNUAIRE FORUM LOOKUP - FORMATION",
TELIS Systèmes & Communications, 13 juillet 1995.

[FORU-95h] EQUIPE FORUM LOOKUP,
"SOLO - FORMATION",
TELIS Systèmes & Communications, date inconnue.

[HEUS-89] B. HEUSE,
INTRODUCTION TO DIRECTORY SERVICES,
Mémoire de fin d'études,
FUNDP - Institut d'Informatique, juin 1989.

[HUIT-94] C. HUITEMA, P-A. PAYS, A. ZAHM ET A. WOERMANN,
"SIMPLE OBJECT LOOK-UP PROTOCOL (SOLO)",
Internet Draft (work in progress), 4 octobre 1994.

[OPEN-92] OPEN SOFTWARE FOUNDATION,
INTRODUCTION TO OSF DCE,
Prentice Hall, New Jersey, 1992.

[RFC-1308] C. WEIDER ET J. REYNOLDS,
"EXECUTIVE INTRODUCTION TO DIRECTORY SERVICES
USING THE X.500 PROTOCOL",
Request for Comments 1308, mars 1992.

[RFC-1309] C. WEIDER, J. REYNOLDS ET S. HEKER,
"TECHNICAL OVERVIEW OF DIRECTORY SERVICES
USING THE X.500 PROTOCOL",
Request for Comments 1309, mars 1992.

[RFC-1484] S. HARDCASTLE-KILLE,
"USING THE OSI DIRECTORY TO ACHIEVE USER FRIENDLY NAMING",
Request for Comments 1484, juillet 1993.

[RFC-1485] S. HARDCASTLE-KILLE,
"A STRING REPRESENTATION OF DISTINGUISHED NAMES",
Request for Comments 1485, juillet 1993.

[RFC-1491] C. WEIDER ET R. WRIGHT,
"A SURVEY OF ADVANCED USAGES OF X.500",
Request for Comments 1491, juillet 1993.

[RFC-1779] S. KILLE,
"A STRING REPRESENTATION OF DISTINGUISHED NAMES",
Request for Comments 1779, mars 1995.

[RFC-1781] S. KILLE,
"USING THE OSI DIRECTORY TO ACHIEVE USER FRIENDLY NAMING",
Request for Comments 1781, mars 1995.

[ROSE-93a] M. ROSE,
THE LITTLE BLACK BOOK,
Prentice Hall, New Jersey, 1993.

[ROSE-93b] W. ROSENBERRY, D. KENNEY ET G. FISHER,
COMPRENDRE DCE,
Addison-Wesley France, mai 1993.

[SUNS-91] SUNSOFT,
"SUNOS MULTI-THREAD ARCHITECTURE",
USENIX, Dallas, hiver 1991,
`http://sunsystem3.informatik.tu-muenchen.de/Solaris/WhitePapers/multi_thread.ps`.

[SUNS-92a] SUNSOFT,
"BEYOND MULTIPROCESSING... MULTITHREADING THE SUNOS KERNEL",
USENIX, San Antonio, juin 1992,
`http://www.sun.com/sunsoft/Developer-products/sig/threads/papers/beyond_mp.ps`.

[SUNS-92b] SUNSOFT,
"IMPLEMENTING LIGHTWEIGHT THREADS",
USENIX, San Antonio, juin 1992,
`ftp://sunsite.sut.ac.jp/pub/sun-info/sun-us/development-tools/multi-threaded/impl_threads.ps`.

[SUNS-92c] SUNSOFT,
"REALTIME SCHEDULING IN SUNOS 5.0",
USENIX, hiver 1992,
`http://www.sun.com/sunsoft/Developer-products/sig/threads/papers/rt_sched.ps`.

[SUNS-93a] SUNSOFT,
"INTERFACES AND THREADS : A TAXONOMY AND GUIDELINES",
12 octobre 1993,
`http://www.sun.com/sunsoft/Developer-products/sig/threads/MT-safe.ps`.

[SUNS-93b] SUNSOFT,
SUNOS 5.2 GUIDE TO MULTI-THREAD PROGRAMMING,
Sun Microsystems, California, mai 1993.

[SUNS-95] SUNSOFT,
"HOW TO BUILD MT SAFE LIBRARIES",
27 septembre 1995,
http://www.sun.com/sunsoft/Developer-products/sig/threads/MTsafe-howto.html.

[SUNS-96] SUNSOFT,
"WRITING MULTITHREADED CODE IN SOLARIS",
Sun Microsystems, California, janvier 1996,
http://www.sun.com/sunsoft/Developer-products/sig/threads/papers/writing_mt_code.ps.

[TS&C-96] DIVISION TELIS SYSTEMES & COMMUNICATIONS,
"WELCOME TO TELIS SYSTEMES & COMMUNICATIONS !",
Page http://www.telis-sc.fr:80/index.html et suivantes.

[WAUG-94] A. WAUGH,
"X.500 AND THE 1993 STANDARD",
Technical Report TR-SA-94-03 (Draft),
CSIRO Division of Information Technology, 16 mars 1994.

Index